Universidad
Carlos III de Madrid

Departamento de
Ingeniería Telemática

PROYECTO FIN DE CARRERA

DESARROLLO DE APLICACIONES COLABORATIVAS MULTIPLATAFORMA: GOOGLE APP ENGINE Y GOOGLE CLOUD ENDPOINTS

Autor: David Jesús Torralbo Gutiérrez
Director: Isaac Seoane Pujol
Tutor: Ignacio Soto Campos

Título: DESARROLLO DE APLICACIONES COLABORATIVAS MULTIPLATAFORMA: GOOGLE APP ENGINE Y GOOGLE CLOUD ENDPOINTS

Autor: David Jesús Torralbo Gutiérrez
Director: Isaac Seoane Pujol
Tutor: Ignacio Soto Campos

EL TRIBUNAL

Presidente: Iván Vidal Fernández

Secretaria: Iria Manuela Estévez Ayres

Vocal: Raquel Pérez Leal

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 19 de Febrero de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIA PRESIDENTE

Agradecimientos

…que eres cero,
que eres cuatro,
que eres hielo;
que tienes dos carros y tres carromatos,
que falta la dicha, de un día, de un año,
de un gozo,
que quien ayer te quiso hoy te escuece
y no tienes quien se atreva
a declarar que el beso, que el luto, que la cuna,
son espectros de pequeñas realidades
y no muertos de carrozas sin domar.

Miguel Ángel Torralbo Barea

Resumen

Este proyecto aborda el Análisis de las diferentes soluciones disponibles para el desarrollo de una aplicación móvil que pueda ejecutarse sobre diferentes plataformas (Android, HTML, iOS, etc) y cuyos datos se sincronicen en la nube mediante tecnologías '*Cloud Computing*' abiertas existente.

En el presente documento se detalla el estudio y definición del proceso a seguir para el uso de la solución elegida (*Google App Engine y Google Cloud Endpoints*) y se analiza el desarrollo de una aplicación de ejemplo que ayude a ilustrar y complementar el estudio y validar los objetivos e hipótesis planteadas.

Palabras clave: Aplicaciones multiplataforma, Cloud Computing, Google App Engine, Google Cloud Endpoints, Android, iOS, HTML5, Channel API, JDO, Business Model Canvas.

Abstract

This project will analyze the different available solutions for the development of mobile applications that can be run on different platforms (Android, HTML, iOS, etc.) which information are synchronized in the cloud by means of available cloud computing open technologies.

The current document will study and describe the process to follow of the selected solution (*Google App Engine and Google Cloud Endpoints*) and analyzes the development of an example of use in order to illustrate and complement this study and validate the objectives and hypothesis proposed.

Keywords: Cross-platform applications, Cloud Computing, Google App Engine, Google Cloud Endpoints, Android, iOS, HTML5, Channel API, JDO, Business Model Canvas.

Índice general

Índice de figuras

Índice de códigos

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1 Motivación del proyecto

El desarrollo actual de aplicaciones móviles parece inmerso en una batalla entre el desarrollo de aplicaciones nativas (Android, iOS, etc) y el desarrollo de aplicaciones que puedan correr en cualquier dispositivo (HTML5, HTML5 embebido en *wrappers* nativos, etc).

A grandes rasgos, las aplicaciones nativas ofrecen mayor variedad de servicios y mejor integración con las funcionalidades disponibles por cada terminal. Igualmente las plataformas de descarga de aplicaciones nativas (los *Markets*) ofrecen un punto de mayor confianza para los usuarios por lo que las aplicaciones nativas son de acceso más común por parte de los usuarios.

Por su lado, las aplicaciones multiplataforma (HTML5 por ejemplo) ofrecen la posibilidad de que con un desarrollo único se pueda acceder a diferentes tipos de dispositivos. Muchas de ellas se ayudan de wrappers nativos para aumentar las funcionalidades disponibles así como una mejor integración con el dispositivo. Así mismo, les posibilita estar presente en los *markets* más importantes de aplicaciones nativas por su mayor cercanía con el usuario, aprovechando la confianza y costumbre del usuario a acceder a este tipo de aplicaciones.

La aparición del concepto de *La Nube* (*Cloud Services*) incrementa la dificultad en la elección de la plataforma en la que desarrollar y el diseño de la misma. La decisión de cómo y dónde emplazar la lógica de negocio para que se ajuste a los requerimientos del sistema elegido así como a las funcionales que se desean presentar, adaptándose a las ventajas que *la nube* ofrece en cuanto a la integración entre diferentes dispositivos de un mismo usuario (ordenador personal, pc en la oficina, teléfono, *tablet*, etc) así como entre diferentes usuarios que hagan uso de funcionalidades de integración (un panel común donde los usuarios interaccionan y ven las acciones del resto, por ejemplo), es uno de los problemas de mayor envergadura al que enfrentarse cuando se diseña la solución técnica. Esta decisión marcará el desarrollo, la tecnología y el equipo técnico a usar para la entrega final del proyecto.

Un proyecto más completo puede incluir acceder a todos los dispositivos disponibles en el mercado: Android, iPhone y iPad (iOS), BlackBerry (HTML5 embebido), Firefox OS (HTML5 embebido), WEB (HTML), etc.

Esta situación puede llevar a diversos problemas que van desde el duplicado de lógica de negocio (hay que desarrollar las mismas funcionalidades para cada una de las plataformas) a la imposibilidad de comunicación entre plataformas (por uso de bases de datos locales).

Con este estudio se pretende facilitar la elección de la plataforma basándose en las siguientes características:

- Multiplataforma: La solución debe dar servicio al mayor número de dispositivos posible.
- Escalabilidad: Debe ser lo suficientemente flexible para poder soportar nuevos dispositivos con el menor impacto sobre el desarrollo.
- Simplicidad: Evitando duplicidad de lógica y reduciendo el tiempo de desarrollo.

La solución más apropiada sería desconectar toda la lógica de negocio del desarrollo de las aplicaciones. Una única lógica de negocio que estuviese accesible desde las diferentes plataformas. Reduciendo y así mejorando por especialización el desarrollo de las aplicaciones de las que se hará uso. Centralizando igualmente la información que se manejará en una única DB que automáticamente se convierte en transparente para la plataforma de la aplicación.

Por lo tanto este estudio se centra en encontrar la plataforma más conveniente y que ofrezca mejores prestaciones para el desarrollo de esa única y centralizada lógica de negocio, y así igualmente estudiar los diferentes métodos de acceso desde cada una de las aplicaciones.

1.2 Objetivos

El objetivo principal de este proyecto es encontrar las tecnologías, las plataformas y el diseño más apropiado para facilitar el desarrollo de aplicaciones que puedan correr bajo el mayor número de plataformas móviles posibles, mejorando las prestaciones que una aplicación en sí puede ofrecer para un dispositivo concreto y optimizando los parámetros del desarrollo de dicha aplicación en cuanto a tiempo, costes y especialización.

La facilidad para el acceso a internet desde un dispositivo móvil a día de hoy junto con la cotidiana situación de poseer más de un solo dispositivo hace pensar que la solución debe ser aquella que además aporte intercomunicación entre los dispositivos. Este dibujo también puede aprovecharse para mejorar las funcionalidades a ofrecer en el diseño de la aplicación, ya que se puede dar servicio no solo a distintos dispositivos de un mismo usuario si no que se podría ofrecer interacción entre diferentes usuarios que hagan uso de una misma aplicación.

Estudiar los actuales canales y soluciones para esta comunicación es la base de este proyecto. Se deben tener en cuenta desde las soluciones más propietarias, tales como gestionar manualmente estas conexiones desde una solución 100% particular, hasta el estudio de los actuales *frameworks open source* que la industria nos puede ofrecer.

La elección de esta plataforma dependerá de factores tales como el tiempo necesario para su implementación, el coste de creación, desarrollo y mantenimiento posterior, la robustez de la plataforma, etc.

Un concepto muy importante a tener en cuenta es la duplicidad de la lógica de negocio. La comunicación entre dispositivos puede ser punto a punto, lo que hace que toda la lógica de negocio se desarrolle en el propio terminal. Si para este caso contamos con dispositivos que corren sobre plataformas diferentes, tales como Android e iOS, esta lógica deberá desarrollarse para ambas versiones. En este caso podríamos estar incurriendo en un situación de duplicidad de código.

Otro de los factores más importantes a contemplar es la optimización de la aplicación en cuanto a desarrollo y mantenimiento. Conseguir un diseño que nos ayude a paliar un problema de duplicidad de lógica de negocio ayudará enormemente a mejorar estos parámetros. El tiempo en el diseño y en el desarrollo de la aplicación se verá reducido, los puntos de riesgo de errores (*bugs*) serán menores y la robustez para enfrentarse a futuros cambios será mayor.

Teniendo en cuenta aspectos como el uso de las conexiones mediante internet y la optimización del código evitando duplicidad, es fácil pensar que una de las soluciones que más encaja en este modelo puede ser la de *la nube* (*Cloud Computing*).

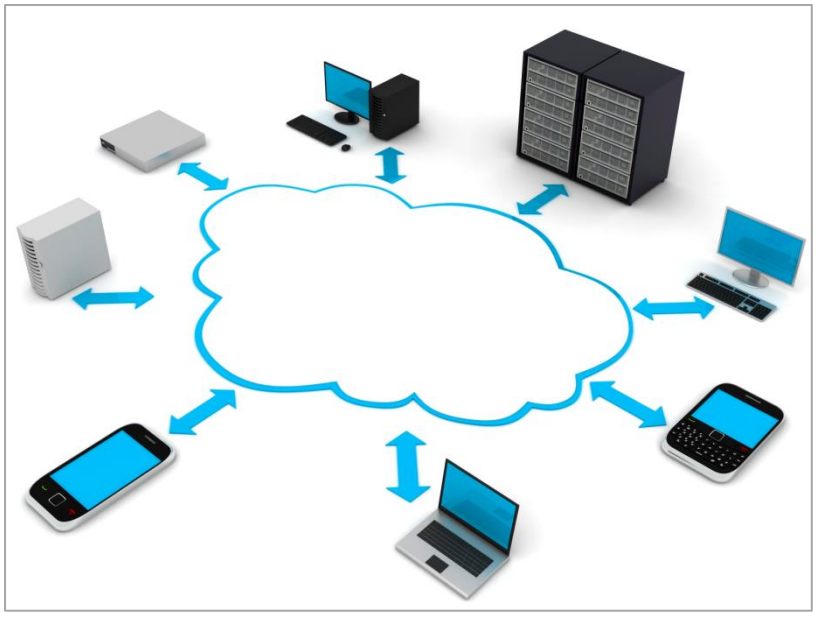


Figura 1. Distribución de dispositivos en la nube

No solo hay que atender a la solución en el servidor o a la gestión de los datos, también hay que prestar atención a cómo estos servicios se integran a los clientes que correrán en los dispositivos móviles. Idílicamente se buscará localizar la mayor parte de lógica de negocio y los datos que se manejan en un único servidor simplificando por tanto el desarrollo de las aplicaciones clientes, en este caso tanto aplicaciones nativas (Android o iOS) como aplicaciones web (HTML5 y HTML5 embebido).

Podemos ver como la solución a priori más óptima va encajando con el patrón *Modelo Vista Controlador* (*MVC*):

- Modelo:
  - Contiene el núcleo de la funcionalidad (dominio) de la aplicación.
  - Encapsula el estado de la aplicación.
  - No sabe nada / independiente del Controlador y la Vista.

- Vista:
  - Es la presentación del Modelo.
  - Puede acceder al Modelo pero nunca cambiar su estado.
  - Puede ser notificada cuando hay un cambio de estado en el Modelo.

- Controlador:
  - Reacciona a la petición del Cliente, ejecutando la acción adecuada y creando el modelo pertinente.

Por último, la persistencia de los datos es igualmente importante en el diseño de la plataforma. Es necesario ver las diferencias entre los distintos modelos existentes (*SQL, NoSQL, Object persistence*, etc) pero se ha de tener en cuenta la relación con los sistemas elegidos (lenguaje de programación, servidor para el controlador, etc) buscando un equilibrio entre la robustez y la complejidad para el proyecto.

1.3 Estructura del proyecto

Teniendo en cuenta todos estos factores y la comparativa actual de servicios disponibles tanto para soluciones propietarias como mediante el uso de componentes *open source* este proyecto se enfoca principalmente en el estudio de la Tecnología *Google Cloud EndPoints*. Esta tecnología nos permite una integración completamente natural de todos los componentes que una aplicación basada en servicios localizados en la red puede necesitar.

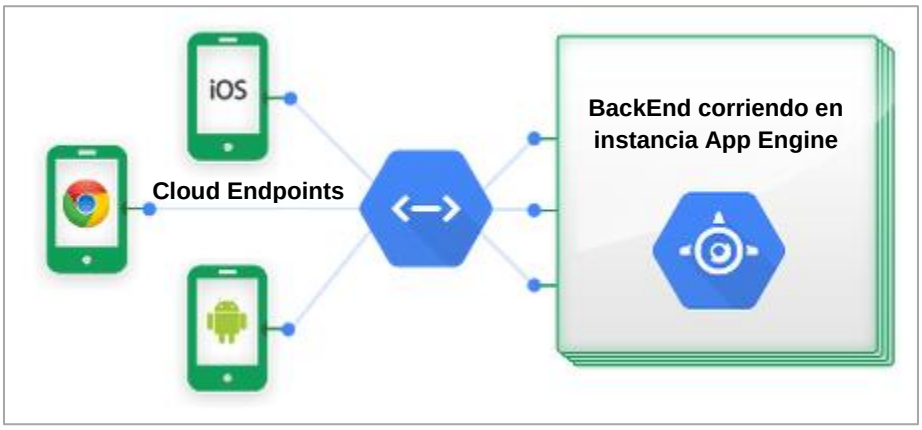


Figura 2. Estructura de Google Cloud Endpoints

En el diseño final, siguiendo los requisitos de esta tecnología, contaremos con los siguientes componentes:

- Servidor de aplicaciones: *Google App Engine* (*GAP*) correrá la lógica de negocio de nuestras aplicaciones. Pool de servidores proporcionado, mantenido y gestionado por Google que ofrece alta calidad en cuanto a disponibilidad, latencia y robustez frente a fallos en la red.

- Lenguajes de programación: *JAVA* como lenguaje único en la parte del servidor así como el lenguaje para los clientes en *Android* y *HTML5* para los clientes en formato Web (*Web Applications*).

- Bases de Datos: *Java Data Objects (JDO)* es la especificación utilizada para almacenar los datos del usuario y de la aplicación. Proporciona gestión automatizada de los datos por parte del servidor, transparencia en el mantenimiento y configuración de la base de datos y una integración completamente familiar con el lenguaje elegido para el servidor.

- Comunicación entre clientes y servidor: *Google Cloud Endpoints* proporciona a su vez las librerías cliente encapsuladas para una perfecta integración con los 3 clientes elegidos creadas a partir de la lógica desarrollada en el servidor.

Además, se ha decidido utilizar metodologías *Agile* para el desarrollo interno del proyecto, definición y toma de requisitos del ejemplo a implementar, seguimiento del tiempo empleado, etc. Específicamente se ha utilizado la metodología *Scrum* y sus mecanismos de ágiles para este proyecto. Así este documento se estructura de la siguiente forma:

1. INTRODUCCIÓN: Se realiza una breve explicación de la motivación de este proyecto así como los objetivos deseados con el mismo. Se analiza a alto nivel la lógica seguida para el diseño de la arquitectura diseñada y se introduce la tecnología finalmente propuesta como solución al problema planteado.

2. ESTADO DEL ARTE: En este apartado se estudia la situación de las tecnologías presentes y se analiza la conveniencia del uso de una tecnología sobre las demás para cada una de los componentes involucrados en el diseño de la solución final.

3. DESARROLLO DEL PROYECTO: Análisis detallado del desarrollo de una aplicación ejemplo básica que haga uso de los componentes finalmente elegidos como tecnologías a usar en este estudio. Se aborda este apartado como un tutorial tipo a seguir para la perfecta integración de todas las partes involucradas en el diseño final. Se contemplarán igualmente posibles problemas que pueden aparecer y su posible solución en cada caso.

4. TIEMPO Y PRESUPUESTO: Informe detallado del tiempo invertido en casa fase del proyecto así como las metodologías utilizadas para la realización del mismo. Se incluye además una estimación del coste económico que conllevaría el desarrollo de un proyecto similar al estudiado en este proyecto de fin de carrera.

Capítulo 2

Estado del Arte

En este capítulo se dividen los distintos componentes que forman las posibles soluciones disponibles y se analiza el estado actual de las tecnologías que podrían utilizarse en cada caso.

2.1 Cloud Computing

2.1.1 Introducción

El desarrollo más sencillo de una aplicación móvil se basa en los componentes locales al dispositivo. Un diseño que contempla la lógica de negocio y la gestión de los datos de forma local, sin comunicación con posibles componentes exteriores ni con otros usuarios que hagan uso de la misma aplicación.

La evolución de las plataformas que corren las aplicaciones, así como los nuevos dispositivos que cada día salen al mercado y junto a la mejora del servicio de internet para dispositivos móviles abren una nueva y potencial posibilidad en el desarrollo de aplicaciones móviles. El *Cloud Computing* se está convirtiendo en un concepto cada día más arraigado en los servicios de los que hacemos uso, y en una funcionalidad extra que los diseñadores y desarrolladores de aplicaciones convierten cada día más en la piedra angular sobre la que apoyar los servicios a implementar.

Parece obvio pues enfocar este proyecto de fin de carrera a un diseño distribuido que pueda hacer uso del mayor número de funcionalidades disponibles que enriquezcan los servicios ofrecidos. El uso del *Cloud Computing* abre además la posibilidad de desarrollar funcionalidades propias de este paradigma a las cuales no se podría tener acceso mediante el desarrollo de aplicaciones completamente locales al terminal.

2.1.2 Definición

Se conoce como *Cloud Computing* o '*La Nube*' al paradigma que define la arquitectura de un proyecto en el que los recursos disponibles son prestados como servicios accesibles a través de una red de comunicaciones, típicamente internet [1]. Define el uso de recursos distribuidos o compartidos emplazados en *pools* virtuales o físicos sobre el uso de recursos locales o personales.

Una de las características a destacar en la implementación de este paradigma es la flexibilidad del servicio prestado. Normalmente esta ventaja viene relacionada con la escalabilidad de la infraestructura, mediante la cual se busca ofrecer la posibilidad de habilitar y deshabilitar recursos disponibles basándose en la demanda que el propio servicio tenga en cada momento. Con esta funcionalidad se consigue que el coste del servicio contratado se ajuste a las necesidades concretas de la aplicación, evitando así ineficiencias y gastos extra por recursos innecesarios.

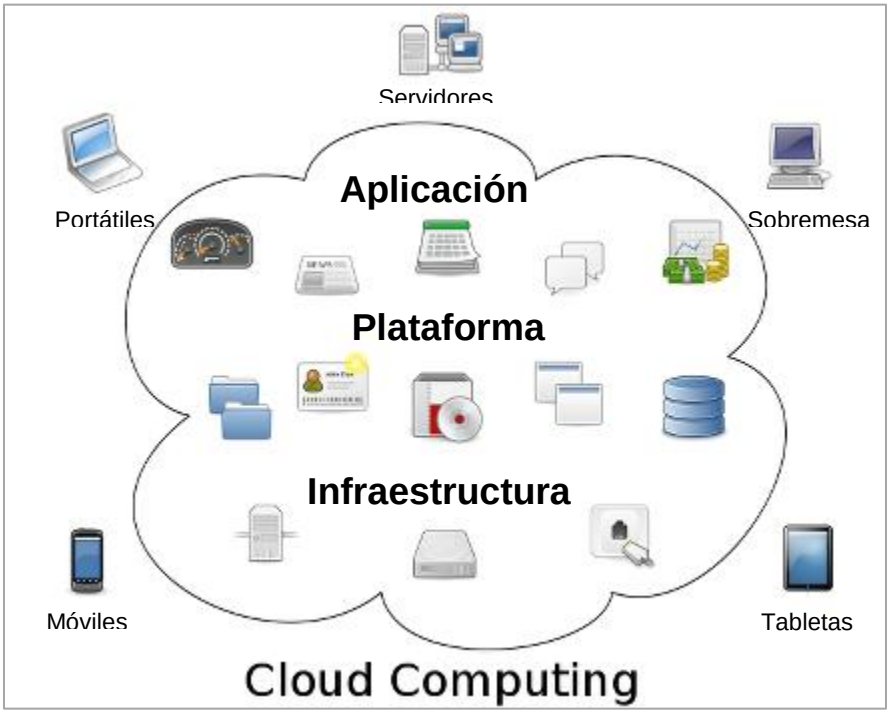


Figura 3. Estructura de Google Cloud Endpoints

Unos de los aspectos más destacables de esta arquitectura son la seguridad y el mantenimiento ofreciendo diversas clasificaciones del servicio a la hora de decidir el diseño y configuración a utilizar [2].

- *Public Cloud*: en este diseño la infraestructura y los recursos de los servicios están localizados y gestionados por una entidad externa o *Cloud Provider*. El acceso a dichos servicios se hace mediante redes públicas, tales como internet. Normalmente los servicios comparten recursos como servidores o redes optimizando así la infraestructura ofrecida por los *providers* aunque los usuarios no son conscientes de los servicios que también corren en los recursos que comparten. El uso de este diseño aporta beneficios como una potente escalabilidad, costes competitivos, modelos de negocios adaptativos, gran fiabilidad, amplia flexibilidad y localización independiente de la infraestructura.

- *Private Cloud*: en este caso la infraestructura y los servidores son mantenidos y localizados en una red interna privada a la organización. La característica más importante de este diseño es la seguridad ya que el acceso público está controlado sino prohibido. Aquí el parámetro del coste es sacrificado frente a la seguridad. Algunos de sus beneficios son claramente la alta seguridad y privacidad de los servicios, mayor control sobre la infraestructura, eficiencia energética, mejora de la fiabilidad del sistema y la gran flexibilidad para migrar servicios no críticos a una *public cloud* con lo que beneficiarse de su diseño.

- *Hybrid cloud*: aquí se combinan el diseño de public cloud destinados para servicios no críticos con el diseño de private cloud para aquellas funcionalidades más sensibles y críticas. Con este modelo se consigue una configuración más personalizada de la infraestructura gracias a la private cloud aprovechándose de los costes más competitivos que ofrecen las public clouds.

2.1.3 Evolución histórica

El primer concepto de *cloud computing* data de los años 50 cuando los conocidos como *mainframe computer* o aquellos sofisticados ordenadores orientados a uso gubernamental o militar se abrieron para el uso por parte de corporaciones e instituciones públicas tales como universidades, buscando optimizar el coste de uso de estos dispositivos mediante el uso compartido tanto del acceso a la *CPU* como a los recursos físicos de los servidores.

Ya en los años 60 el prestigioso informático norteamericano John McCarthy predijo que "*computation may someday be organized as a public utility*" o en otras palabras, la informática y sus aplicaciones serían algún día ofrecidas como un uso completamente público.

Pero fue el canadiense Douglas Parkhill el pionero en el estudio del *cloud computing* quien exploró a fondo las necesidades y beneficios de esta arquitectura, publicando en 1966 el libro "*The Challenge of the Computer Utility*".

Durante este tiempo grandes corporaciones se unieron a la idea de ofrecer parte de sus infraestructuras buscando así optimizar sus costes. Tales como SBC (*Service Bureau Corporation* en 1957) proveniente de *IBM*, *Tymshare* en 1966, National CSS en 1967, etc.

Es ya en los años 90 cuando las compañías de telecomunicaciones comienzan a cambiar su oferta de modelo de servicios dedicado punto-a-punto en otros modelos más dinámicos basados en redes virtuales (VPN) con las que podían ofrecer la misma calidad de servicio pero a un coste mucho más competitivo. Esta fue la base para la expansión del *cloud computing*, desde la cual se mejoraron los servicios prestados cubriendo incluso más allá de la idea inicial de la infraestructura de las redes y se empezó a dar importancia a estos modelos, poniendo atención por ejemplo a los algoritmos utilizados cuyas mejoras provocarían optimización en las infraestructuras, plataformas y aplicaciones ofrecidas.

A partir del año 2000 y con *Amazon* como principal estandarte el *cloud computing* se conseguirá uno de sus mayores avances cuando por fin se decide invertir plenamente en la mejora de sus *data center*. Hasta ahora el uso convencional de estos mismo apenas ascendía el 10%, pero con este avance se vio como era posible mejorar considerablemente la eficiencia de los servicios. Fue entonces en el año 2006 cuando *Amazon* lanzó su conocido servicio *Amazon Web Services* (*AWS*), base de futuros proveedores tales como *Eucalyptus* u *OpenNebula* en el 2008.

Google también ve en el *cloud computing* ventajas tanto estructurales como comerciales y decide lanzar una primera versión inicial allá por Abril del 2008 manteniéndose en una constante evolución del producto (casi mensual) llegando a la actual 1.8.9 de Enero del 2014.

Actualmente, y más allá de los innumerables proveedores de *cloud computing* de los que se disponen y de las capas de servicio que ofrecen cada uno, parece ser que *Amazon* y *Google* constituyen los dos ejemplos más característicos de este modelo de comunicación. Sus enormes infraestructuras físicas en forma de *data centers* así como el amplísimo abanico de servicios que ofrecen bajo sus plataformas unidas a los ambiciosos *SLA (Service Level Agreement)* a los que se comprometen hacen de estos dos proveedores los referentes principales de estudio en este proyecto.

2.1.4 Clasificación por capas

El concepto de *cloud computing* abarca desde el acceso del cliente a los servicios y aplicaciones ofrecidas por los proveedores, hasta los niveles más bajos de la infraestructura de comunicaciones. Este ámbito puede resultar muy amplio y por ello se decidió agrupar las características de los recursos disponibles creando la siguiente división de servicios por capas.

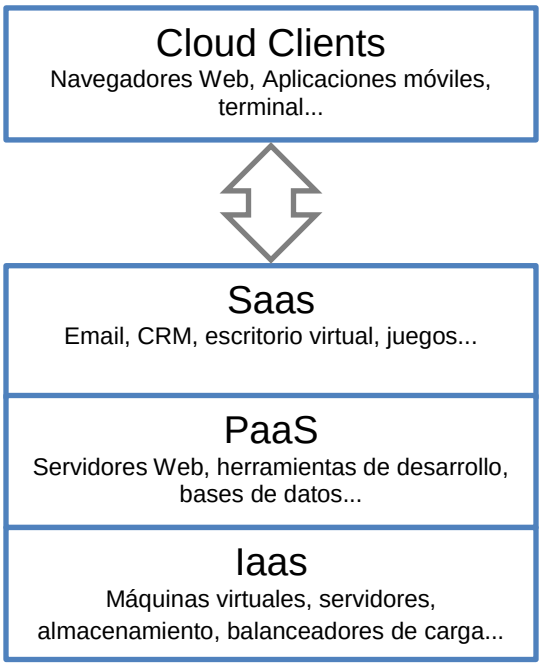


Figura 4. Clasificación por capas de la nube

Esta clasificación ayudó también a identificar los modelos comerciales a ofrecer así como la especialización de empresas la cuales se apoyan unas en otras para poder ofrecer multitud de soluciones que pueden ser más o menos complejas dependiendo de las necesidades particulares.

- Infrastructure-As-A-Service (IaaS)

Por definición, este modelo de *cloud computing* ofrece a sus clientes acceso a recursos informáticos a través de redes de comunicación, habitualmente internet, pero en este caso estos recursos están compuestos por los niveles más bajos de la comunicación tales como espacio virtual para los servidores, conexión entre redes, ancho de banda, direcciones *IP* y hasta balanceadores de carga. En definitiva todo lo relacionado a la infraestructura de una red desde el punto de vista del *hardware*.

Es el proveedor de este servicio el encargado del mantenimiento del mismo y los clientes los encargados de la creación y mantenimiento de la plataforma sobre esta infraestructura.

Típicamente este servicio es utilizado por compañías que prefieren externalizar el mantenimiento del hardware por su complejidad y coste aprovechándose de los precios tan competitivos del mercado actual.

Entre los beneficios del uso de estos servicios se pueden destacar tales como la *escalabilidad* gracias a la facilidad para aumentar o reducir los recursos contratados, el *coste de uso* basado en el uso bajo demanda evitando sobrecostes por recursos no usados eficientemente, la *independencia de la localización* de los recursos estando disponibles desde básicamente cualquier localización con conexión a internet or la *reducción de los puntos de fallo* del servicio al tener disponibles multitud de recursos en paralelo para subsanar dicho fallo.

Aunque como siempre discutible, la clasificación actual del top 5 de proveedores de *IaaS* podría ser según *Networkword* [3].

1. *Amazon Web Services*: con su servicio *Elastic Cloud 2* (*EC2*) es el líder indiscutible en esta materia incrementando la variedad de servicios a ofrecer continuamente.

2. *Bluelock*: gran alternativa para creación de *private, public and hybrid clouds* gracias a su servicio *Bluelock Virtual Data Center*, manejado por un potente panel con el que sus usuarios pueden fácilmente entender la estructura de sus servicios y cuales conllevan el mayor coste en cada momento.

3. *CSC*: llegado a este mercado solo hace 3 años con una inversión bastante potente consiguiendo recientemente un gran posicionamiento por su servicio *VCE IaaS*.

4. *GoGrid*: reconocidos por ser una *pure-play cloud company*, sus servicios son aplicables tanto a *private clouds* como a *public clouds*. *GoGrid Exchange* es su servicio más importante.

5. *IBM*: con una reciente y ambiciosa inversión se consolida como una de las empresas más potentes del mercado. *SmartCloud Enterprise* y *SmartCloud*+ son sus señas de identidad más reconocibles.

- Platform-As-A-Service (PaaS)

En esta clase de servicio, los proveedores ponen a disposición de sus usuarios la plataforma y los entornos necesarios para que los mismos desarrolladores puedan desplegar sus aplicaciones a través de internet.

Los proveedores suelen ofrecer a los clientes un amplio abanico de diferentes funcionalidades que pueden ser contratadas para ser usados por sus aplicaciones. Estas configuraciones pueden ir desde básicas hasta para usuarios muy avanzados. Estos servicios están en constante actualización apareciendo nuevas funcionalidades y mejorando las ya existentes, todo servido por el mismo proveedor.

El modelo de negocio está basado en la contratación de los servicios bajo demanda, haciendo uso solo de aquellos que sean de interés para el usuario. Esto ofrece a los clientes un gran beneficio económico gracias a que los recursos están continuamente compartidos dentro de la infraestructura del proveedor.

La lista de funcionalidades ofrecidas puede variar pero la lista más común podría ser:

- Sistema operativo y sistema de bases de datos.
- Entorno de *scripting* en el servidor.
- Almacenamiento y *Hosting*.
- Soporte.
- Acceso a internet.
- Herramientas de soporte al diseño y al desarrollo.

Como beneficios se pueden destacar que no es necesario la inversión en infraestructuras físicas ya que son proveídas por el mismo proveedor, facilidad en el despliegue de aplicaciones sobre los servidores que correrán las mismas, flexibilidad en la configuración de las funcionalidades a usar en cada momento, seguridad a cargo del proveedor tales como *backups* o incluso la facilidad para el trabajo con equipos de trabajo en remoto. En esta sección es posible destacar como el top 5 de los proveedores de *PaaS* a:

1. *Google*: con su multilenguaje plataforma *Google App Engine* sobre la que se puede desplegar aplicaciones y servicios basados en *JAVA, Python, PHP and Go.* Con una extensísima comunidad de desarrolladores y un sin fin de servicios disponibles para ser usados por las aplicaciones.

2. *Amazon*: su *AWS EC2* es sin duda uno de los servicios *PaaS* más potentes del mercado con una mención especial a su plataforma para el despliegue de aplicaciones (*AWS Elastic Beanstalk*) que facilita significativamente el trabajo de los desarrolladores.

3. *Windows*: *Windows Azure* es el servicio proporcionado que aunque con apenas algo más de un año de vida ya constituye una de las plataformas más completas, incluyendo no solo el despliegue sino hasta *hosting* e incluso servicio de mantenimiento.

4. *Salesforce*: con su servicio *Forces* has conseguido abrir su propia infraestructura para ser utilizada desde aplicaciones externas aprovechándose de su sistema de despliegue sobre la plataforma *salesforce*.

5. *ThinkGrid:* donde abarcan básicamente todos los aspectos del *cloud*. Mediante esta plataforma es posible diseñar, desarrollar y lanzar servicios comerciales.

- Software-As-A-Service (SaaS)

Modelo de *cloud computing* en el que los usuarios pueden acceder a aplicaciones finales a través de internet. Estas aplicaciones están hosteadas en *la nube* y están disponibles para cualquier cliente o corporación mediante cualquier dispositivo habilitado para navegar por internet.

El modelo *SaaS* ofrece a sus clientes un acceso a los servicios en forma de suscripción, normalmente mensual, frente un modelo tradicional de compra del software. Además es destacable el hecho de que el uso de la aplicación se *online* con almacenamiento en *la nube* de cualquier información y ficheros generados.

Los beneficios más destacables de este modelo van desde el coste ya que no es necesario ninguna inversión en elementos de hardware por parte del cliente ni coste por configuración inicial del servicio, compatibilidad *cross-device* pudiéndose hacer uso del servicio desde diferentes dispositivos, acceso a los mismos desde cualquier localización, funcionalidades actualizadas automáticamente sin implicación del cliente, uso escalado según las necesidades del usuario con posibilidad de incrementar el mismo mediante la actualización de los servicios contratados, etc.

2.1.5 Servidores de aplicaciones

- Google App Engine (GAE)

PaaS ofrecido por *Google* en el que los desarrolladores pueden desplegar sus aplicaciones *WEB* dentro de la infraestructura de servidores de *Google* [4]. Google *App Engine* ofrece entre sus servicios una potente escalabilidad que dependerá del número de peticiones que cada servicio obtenga, aumentando así el número de servidores disponibles en cada momento.

Servicio multiplataforma aceptando desarrollos basados en los lenguajes *JAVA, Python, PHP y Go*, es lanzado por primera vez en Abril del año 2008. Desde entonces y como puede verse en su historial de versiones los desarrollos y actualizaciones han sido constantes hasta la actualidad.

Para un estudio más detallado es posible consultar el Anexo I. Estudio del Arte: *Google App Engine*, adjunto en este proyecto.

- Amazon EC2

Pionero y uno de los proveedores *PaaS* más conocidos y robustos del mercado, ofrece una completa infraestructura para el despliegue de aplicaciones que serán accedidas a través de internet.

En Agosto del año 2006 se lanzó la primera versión beta [5] y desde entonces ha estado en continuo crecimiento incorporando a su lista de servicios importantes funcionalidades como:

- Configuración escalable y flexible.
- Control total sobre las instancias.
- Diseñado especialmente para trabajar conjuntamente con otros servicios de *Amazon* como *Amazon S3* (para almacenamiento), *Amazon RDS* (bases de datos relacionales) o *Amazon SQS*.
- Gran fiabilidad con hasta un 99,95% de disponibilidad de los recursos según su *SLA*.
- Seguridad basada en la infraestructura propia de *Amazon*.
- Precios competitivos ofreciendo diferentes tipos de opciones como *On-Demand* o *Reserved Instances*.
- Facilidad para la inicialización en los servicios.

Se adjunta a este proyecto el Anexo II. Estudio del Arte: *Amazon EC2*, en el que se detallan las especificaciones de este servicio.

2.1.6 Comunicación cliente - servidor

El objeto de este proyecto es analizar y encontrar la metodología y herramientas más óptimas para la creación de una aplicación multiplataforma optimizando los recursos a utilizar y los recursos necesarios para futuras mejoras o ampliaciones. Parece claro que el *cloud computing* es la solución que más se acerca a estas necesidades emplazando la lógica de negocio en un servidor común.

Esta arquitectura requiere de mecanismos de comunicación entre el servidor y los diferentes clientes que consumen estos servicios. Existen multitud de tecnologías con las que solventar este problema, las más características son las siguientes.

- JAVA Servlets

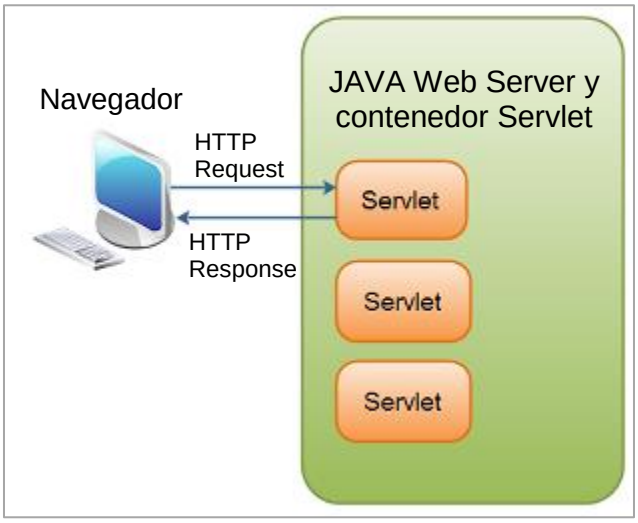


Figura 5. Java Servlets

Tecnología *JAVA* por excelencia utilizada para extender las funcionalidades de un servidor *Web* con la cual se puede crear un servicio *Web* sin la necesidad de la vista. Una de las características más importantes es que no está ligada a ninguna plataforma ni tipo de servidor, lo que deja gran libertad de elección de los mismos.

La comunicación se basa en peticiones *HTTP* que ejecutarán la lógica asociada a cada recurso pedido. El cliente deberá entonces entender la respuesta y completar la lógica desde la aplicación cliente. En un uso más genérico de los *JAVA Servlets* la respuesta puede ser en formato *HTML* con lo que directamente visualizar el resultado, pero es posible crear esta respuesta usando otros formatos tales como *XML*, que sí podría ser parseado y manejado por los clientes externos.

La evolución natural de esta tecnología desde el punto de vista de los servicios *Web* son las *JSP* (*Java Server Pages*) en las que básicamente es el código *JAVA* es el que es embebido dentro de código *HTML*. En este caso los *JSP* son transformados en servlets en tiempo de ejecución.

- WEB Services (SOAP, REST)

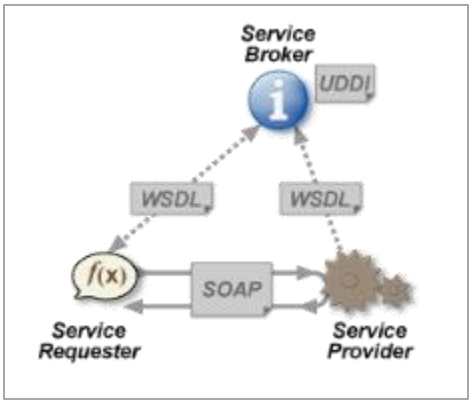


Figura 6. Descripción de un WEB Service

Tecnología utilizada para la comunicación a través de internet entre dos dispositivos informáticos [6]. Es en esencia un servicio disponible en una determinada dirección web que queda a la espera de ser ejecutado por otros recursos de la red. Esta tecnología está esencialmente regulada por el consorcio *W3C (World Wide Web Consortium)* en el que se definen los estándares a utilizar tales como la arquitectura y la reglamentación.

Los *web services* por definición deben ser auto definidos gracias a los descriptores *WSDL (Web Services Description Language)*, que se encargan de describir los detalles relacionados con los servicios que el web service dispone. Se tratan de ficheros en formato *XML* con los que es posible conocer los requisitos funcionales necesarios para realizar la comunicación con el servidor.

Es posible conocer estos *web services* gracias al uso de *UDDI (Universal Description, Discovery and Integration)* cuya funcionalidad básica es la de publicar la información relacionada con los servicios disponibles. Así gracias a este servicio y al descriptor *WSDL* del que dispone, cualquier dispositivo de la red puede acceder y ejecutar la lógica desarrollada por el *web service* usando para ello el protocolo *SOAP (Simple Object Access Protocol).*

Esta arquitectura podría considerarse similar al tradicional modelo Cliente/Servidor de las páginas web, pero difiere principalmente en que el servidor de los *web services* carece de interfaz de usuario. Estos servicios básicamente comparten recursos como los datos, procesos y hasta la lógica de negocio.

Como evolución más destacable de esta arquitectura se pueden encontrar los servicios *REST (Representational State Transfer)*. Concepto originado en el año 2000 por Roy Fielding y actualmente uno de los protocolos de comunicación más populares.

Caracterizado por romper con rigidez de la comunicación *SOAP*, *REST* carece de descriptores de sus servicios proporcionando así una mayor flexibilidad a los mismos. *RESTFul* es el API asociado a este protocolo que se basa en el uso de estándares *HTTP* para la definición de los puntos de acceso a los servicios, tales como *GET, POST, PUT o DELETE* que completarán el modelo de operaciones *CRUD (Create, Read, Update y Delete)*.

De la misma forma la comunicación en ambos sentidos se hace mediante el uso de formatos optimizados para comunicación por internet, tal como *JSON (JavaScript Object Notation)* como formato más común aunque es posible hacer uso de otros como *XML*.

Es importante tener en cuenta aquí que la comunicación se realiza a través de protocolo *HTTP*, operaciones normalmente complejas y críticas para toda aplicación, aún más si cabe para aplicaciones móviles.

De igual importancia hay que reseñar que, en todo caso, la respuesta servida por el *web service* al cliente debe ser maquetada y entendida por éste para una completa ejecución de los servicios. El parseo de las respuestas junto con el más que común desconocimiento del formato y estructura de las mismas supone un punto delicado y complejo para el desarrollo de las aplicaciones cliente.

- Google Cloud EndPoints

Durante una de sus charlas en el *Google I/O* de 2012 y bajo el nombre de "*Building Mobile App Engine Backends for Android, iOS and the Web*" *Google* presenta un nuevo servicio destinado a facilitar el desarrollo de aplicaciones móviles basándose en la idea de centralizar en un solo *backend* toda la lógica de negocio, llamado *Google Cloud EndPoints*.

Lanzado entonces como servicio experimental y con un grupo reducido de desarrolladores como "*true testers*", se convierte en funcionalidad oficial de *Google App Engine* en su versión 1.8.7 en Noviembre de 2013.

Los *Google Cloud EndPoints* están diseñados para facilitar el desarrollo de todos los componentes que una aplicación móvil multiplataforma puede necesitar. Este modelo aúna gran cantidad de las ventajas que los diferentes servicios y modelos que *Google* ofrece. A los servicios ofrecidos por su *Paas Google App Engine* se unen las nuevas herramientas que *Google* incluye en su *SDK* con el que poder generar de forma automática y con un solo click todos estos componentes.

Se define *Google App Engine* como el servidor de aplicaciones que ejecutará la lógica de negocio. El conjunto de herramientas que *Google* ofrece en su *plugin,* desarrollado específicamente para *Eclipse,* facilitan enormemente el trabajo del desarrollador.

Las nuevas funcionalidades introducidas encapsulan las acciones necesarias para el desarrollo de los diferentes componentes necesarios tanto en la lógica de negocio de la misma aplicación como en la generación de las librerías que ayudarán a una comunicación casi transparente desde las aplicaciones clientes al servidor:

- Generación de la lógica de negocio básica a partir de un simple modelo de datos.
- Posibilidad de incrementar la lógica de negocio gracias a anotaciones.
- Definición automática de una API a partir de la lógica de negocio creada.
- Creación de las librerías cliente que serán usadas por las aplicaciones que acceden al servidor.
- Acceso a la *API Console* con la que se puede probar y testear de una forma simple la lógica generada sin necesidad de desarrollar ninguna aplicación cliente para ello.
- Despliegue y publicación automática de los *Cloud EndPoints* en un servidor público *App Engine*.

La tecnología de la que *Google App Engine* hace uso para la lógica de negocio se basa en la especificación *REST*, aprovechando las ventajas que este modelo ofrece y el potencial de su desarrollo. En la generación primaria de los *endPoints*, por ejemplo, se definen los siguiente servicios básicos que hacen uso de los métodos *HTTP* tal y como se especifica:

- listar datos -> GET
- acceder a uno de los datos -> GET
- añadir un dato nuevo -> POST
- actualizar datos -> PUT
- eliminar un dato -> DELETE

A la hora de acceder desde las aplicaciones cliente a los servicios disponibles hay que tener en cuenta diversos aspectos durante la fase de desarrollo de los mismos. Las librerías cliente que los *Google Cloud Endpoints* generan son de gran ayuda para encapsular alguno de los aspectos más importantes de la comunicación con el fin de mejorar el desarrollo y reducir los puntos de error:

- Configuración del *endPoint* al que realizar las peticiones. Este dato está encapsulado en las librerías generadas, por lo que el desarrollador no tiene que prestar atención a este dato. Si este *endPoint* cambia, solo será necesario generar y proveer de nuevo la librería a utilizar.
- Establecimiento, mantenimiento y cierre de la conexión. En este caso las librerías generadas se encargan automáticamente de estas acciones lo cual facilita enormemente el desarrollo de los clientes y reducen los puntos críticos de la aplicación.
- Tratamiento de los datos de la respuesta. En el desarrollo desde aplicaciones *JAVA* la respuesta se presenta en forma del objeto original devuelto por el servicio, con lo cual el tratamiento de la información es mucho más natural e intuitivo.

Con este servicio se crean las librerías necesarias para acceder desde diferentes plataformas móviles, con lo que los desarrolladores, tanto de la parte servidora como de las aplicaciones cliente, pueden centrar sus esfuerzos y tiempos de desarrollo a aspectos más cercanos a la lógica de negocio en sí. Las plataformas a las que da soporte son:

- HTML, mediante las librerías *javascript* que genera.
- Android.
- iOS.

Además, al tratarse de un servicio que trabaja sobre la especificación *REST*, sería posible aprovecharse de este modelo para acceder a los servicios desde plataformas que no estén directamente soportadas por los *Google Cloud EndPoints*.

Esta especificación además soporta modelos de autenticación en el acceso a los servicios, acotando por tanto los clientes que pueden hacer uso de la *API*. Esta autenticación está basada en la especificación *OAuth2*, y especialmente integrada con el modelo *Google+ Sing-In*, mediante la cual es posible autenticar al usuario usando las credenciales que posee en *Google*, tal y como se hace actualmente para cualquiera de los servicios de *Google*.

Incluso, gracias al uso de una especificación estándar como *OAuth2*, sería posible implementar modelos de autenticado proveniente de terceros, tales como *Facebook*.

2.1.7 Bases de datos

La arquitectura de bases de datos a usar es uno de los aspectos del diseño más variable. En la elección de este modelo hay que tener en cuenta diferentes parámetros que podrían ayudar a el diseño de todo el modelo de datos de la aplicación.

Es importante para ello conocer a fondo el negocio para el que la aplicación será desarrollada, eso puede dar ciertos detalles sobre las necesidades de la base de datos, como por ejemplo la carga de datos que deberá soportarse, la flexibilidad y volatilidad de los mismo, si se tratará de accesos a datos críticos en tiempo real, el nivel de seguridad que se quiera implementar, etc.

Desde el punto de vista de una *startup*, también se recomienda tener en cuenta detalles como:

- Siempre se puede invertir algo más de tiempo en busca de la solución ideal.
- Es recomendable implementar la solución más sencilla para los problemas actuales.
- Todo modelo de bases de datos puede ser mejorado o migrado según las necesidades que vayan apareciendo en el crecimiento del negocio.
- Hacerse eco de casos de estudio reales que hayan funcionado exitosamente más allá que de simplemente las especificaciones técnicas de cada modelo.

Las arquitecturas predominantes en la actualidad serían los modelos *SQL y NoSQL*.

- SQL (Structured Query Language)

Este modelo está destinado a bases de datos relacionales, donde la estructura del sistema adquiere una gran importancia y el manejo de los datos se hace de una forma más ordenada y precisa gracias a el formato de queries aceptado. La flexibilidad y potencial de estos sistemas relacionales es la base del lenguaje *SQL*.

Sistema de referencia y más usado desde que en los años 70 se dejaron los antiguos modos de almacenamiento de la información en ficheros. Aplicable a básicamente cualquier diseño, algunas de sus puntos débiles frente a las necesidades de los nuevos modelos de negocio emergentes propiciaron la aparición de un nuevo lenguaje mucho más acorde a estas necesidades, *NoSQL*.

- NoSQL (Not Only SQL)

En este caso, el modelo tradicional relacional es descartado y se aboga más por un modelo no estructurado donde el acceso a datos en tiempo real, por ejemplo desde servicios web, debe estar lo más optimizado posible.

De fácil implementación ya que no requiere un estudio previo para un diseño optimizado de las relaciones entre tablas, goza de gran popularidad entre los desarrollos de servicios críticos en cuanto a tiempos de acceso.

Dada su naturaleza, solo soporta directrices de tipo INSERT, DELETE y SELECT ya que se encuentra optimizado para las operaciones de recuperar y salvar información.

Caracterizado por su escalabilidad y rendimiento, su peculiar sistema de datos distribuido hace que este modelo sea uno de los mejores preparados para la tolerancia ante fallos.

- Java Object Persistence

Dado *JAVA* es uno de los lenguajes con mayor incursión y es posible ser usado como lenguaje de desarrollo en algunos de los posibles componentes a definir en el diseño final, sobre todo en el lado servidor (*Google App Engine o Amazon EC2*), es interesante analizar igualmente este modelo que la especificación *JAVA* ofrece.

Este modelo proporciona un nivel extra de abstracción a los desarrolladores, mediante el cual el acceso a la base de datos se realiza de forma casi transparente. Caracterizado por un diseño de tablas basado en la estructura interna de los objetos sobre los que se quiere aplicar la persistencia, esta especificación hace uso del paradigma de *Orientación a Objetos* para definir la relación entre entidades así como la estructura de la información a almacenar.

Así pues los objetos creados definirán en sí las tablas a crear y sus variables de clase las columnas de las mismas.

Es curioso ver como en este modelo, mediantes sus *APIs*, permite realizar consultas sobre los datos típicas del modelo *SQL*. Y a la vez entender que la estructura de las tablas goza de gran flexibilidad debido a la estructura propia de los objetos en los cuales la información puede ser ciertamente cambiante.

Dentro de esta definición es posible encontrar dos grandes especificaciones, JPA (*Java Persist API*) y JDO (*Java Data Objects*). Es posible consultar más detalles sobre las diferencias entre estos dos modelos en el Anexo III. Estudio del Arte: JDO vs JPA.

En el caso de los servidores Google App Engine, estos dos modelos se encuentran totalmente integrados en su desarrollo proporcionando toda la infraestructura necesaria, tanto desde el punto de vista del desarrollo, como en la configuración de la base de datos así como en la visualización mediante el panel de mantenimiento de los servidores.

2.2 Herramientas de desarrollo

2.2.1 Entornos de desarrollo integrados (IDE)

Los entornos de desarrollo (o Integrated Development Environment) hacen que la programación, usando los diferentes lenguajes requeridos, como el uso de los diferentes *APIs* de los que cada tecnología dispone sean tareas mucho más automáticas, intuitivas y rápidas, facilitando servicios que encapsulan automáticamente diversas tareas y reduciendo los casos de error típicos del desarrollo de aplicaciones.

Algunos de propósito general y otros desarrollados específicamente para algunos de los componentes que se necesitarán, son una herramienta fundamental para un desarrollo óptimo. Complementados con nuevas funcionalidades e incluso haciendo uso de *plugins* desarrollados propiamente para ellos, los diferentes proveedores consiguen que la fase de desarrollo de las aplicaciones se simplifiquen a niveles tales como despliegues a producción con un solo click.

Algunos de los entornos que más pueden encajar con el desarrollo de final de este proyecto son:

- Eclipse

Actualmente, el *IDE* más utilizado en el mundo del desarrollo de software, especialmente bajo el lenguaje *Java*, con una incursión del 65% del mercado. La *Eclipse Open Source Community* es la encargada de regular sus actualizaciones con las que cuenta con más de dos centenares de proyectos para cubrir diferentes aspectos de la herramienta.

Una de las características a destacar es su flexibilidad. Esta herramienta acepta desarrollo de funcionalidades personalizadas que se integran completamente con la interfaz, los llamados *plug-ins*. Esto ha hecho que multitud de compañías y comunidades elijan esta herramienta como base de su desarrollo, generando sus propios servicios que facilitan la integración de sus desarrollos.

Uno de los ejemplos más conocidos de estos *plug-ins* son los proporcionados por *Google* para la integración de sus servicios con la herramienta, consiguiendo así que los desarrolladores puedan ejecutar multitud de acciones en un solo click y desde un solo punto de despliegue. Por ejemplo ayuda al desarrollo sobre el *Paas Google App Engine* con su *plug-in* en el que es posible publicar el proyecto desde el mismo *Eclipse*, o aplicar las funcionalidades de los *Google Cloud EndPoints*, etc.

Otro *plug-in* importante proporcionado por *Google* serían los destinados al desarrollo sobre la plataforma *Android* (tanto para la configuración y uso de su *SDK* como el destinado a la simulación de los dispositivos o *Android Virtual Device Manager*).

Actualmente en su versión estable 4.4.1 (*Luna*), además de su principal lenguaje de desarrollo *Java*, este *IDE* puede ser también usado para el desarrollo sobre otros lenguajes de programación tales como:

- Ada
- C
- C++
- COBOL
- Fortran
- Haskell
- JavaScript
- Lasso
- Perl
- PHP
- Python
- R
- Ruby (including Ruby on Rails framework)
- Scala
- Clojure
- Groovy
- Scheme
- Erlang

Es por tanto uno de los *IDE* más extendidos y usados, proporcionando con ello gran soporte por parte de la comunidad de desarrolladores y, desde el punto de vista del desarrollo sobre los servicios de *Google*, la herramienta base más integrada con su plataforma.

- IntelliJ IDEA

La gran alternativa actual a *Eclipse*, este *IDE* goza de gran popularidad y se encuentra en continuo crecimiento. Con un interfaz realmente amigable y un potente gestor de ayudas al desarrollo se ha convertido incluso en la base para la creación del exclusivo *IDE* destinado al desarrollo *Android* (*Android Studio*).

Este IDE está desarrollado por la compañía JetBrains quien lanzó la primera versión estable fue lanzada en el año 2001 y actualmente se encuentra en su versión 13.0.2 (Build: 133.696) de Enero del 2014.

Este *IDE* cuenta con dos ediciones de distribución, una completamente gratuita y otra de pago. La diferencia fundamental se basa en las funcionalidades disponibles en una y otra edición y en los framework y servidores soportados.

Muy acogido por la comunidad de desarrolladores por la limpieza y sencillez de la interfaz, por sus servicios de ayuda al desarrollo y por el extenso catálogo de lenguajes soportado.

- Android Studio

IDE especializado en el desarrollo de aplicaciones *Android*, basado en el *IDE IntelliJ* de *JetBrains*. Soportado por *Google*, integra las herramientas necesarias para un desarrollo más eficaz e integrado de aplicaciones *Android*.

Fue presentado en una de las charlas del *Google I/O* del año 2013 (*What's new in Android Developer Tools*), actualmente se encuentra en su versión estable 0.4.4 de Febrero del 2014.

Aprovecha las ventajas de *IntelliJ* en cuanto a la interfaz tan amigable junto a los sistemas de ayuda al desarrollo y además añade las especificaciones propias del desarrollo *Android* tales como su sistema de depurado, funcionalidades propias de *Google* (como las relacionadas con los *Google Cloud EndPoints*) y su potente funcionalidad de simulación de *layouts* sobre diferentes configuraciones *Android*.

La comunidad *Android* ha recibido a este *IDE* con gran expectación y el soporte, manuales y ejemplos desarrollados sobre esta plataforma se ha multiplicado desde su lanzamiento.

2.2.2 Sistemas de Control de Versiones

El uso de un sistema de control de versiones es un aspecto muy importante en el desarrollo actual de aplicaciones. Proporciona un repositorio central donde poder sincronizar el código sobre el que se está trabajando, esto permite además la posibilidad de realizar un trabajo colaborativo entre diferentes desarrolladores u organizaciones.

Esta sincronización permite que el código pueda ser accedido desde diferentes localizaciones y desde diferentes dispositivos, consiguiendo así un acceso global al desarrollo en todo momento.

Además este modelo es de gran ayuda para el desarrollo de este proyecto con el que se ha conseguido minimizar el impacto del trabajo en remoto entre alumno y profesor, ya que en todo momento todos las figuras implicadas están actualizadas de los últimos cambios generados en el código.

Algunas de las implementaciones más conocidas de esta metodología son:

- Concurrent Versions System (CVS)

Uno de los sistemas de control de versiones más conocido y utilizado [7]. Actualmente se encuentra en su última versión estable 1.11.23 publicada en el 2008. Su desarrollo se encuentra es una situación tan madura que la comunidad que se encarga de su desarrollo no contempla nueva versiones hasta que aparezcan nuevas funcionalidades o reporten nuevos *bugs*. Hay que tener en cuenta que su primera versión fue publicada en el 1990.

De gran potencia ya que no solo guarda el historial de cambios realizados sobre el código, sino que además guarda un copia del código asociado a cada versión. Gracias a esta funcionalidad es posible recuperar cualquier versión desde el mismo repositorio.

Su popularidad ha hecho que se trate del sistema más extendido incluso en el ámbito corporativo. Existen multitud de desarrollos que facilitan el uso y la integración de este sistema, que van desde *Unix scripts*, aplicaciones de escritorio o incluso integraciones con los mismos entornos de desarrollo como es el ejemplo de *Eclipse*.

- Apache Subversion (SVN)

Este sistema de control de versiones es considerado como el sucesor de *CVS*, una evolución natural que intenta llegar a ser un cliente más compatible que el estable y maduro *CVS* [8].

Basa su fuerza es ser un sistema de control de versiones completamente *open-source* que pretende además solucionar los problemas que *CVS* presentaba. Durante su evolución ha conseguido incluir multitud de nuevas funcionalidades que el anterior sistema no presentaba, mejorando así el trabajo de los desarrolladores en cuanto al control de versiones se trataba.

Debido a ser concebido como la solución a los *bugs* y a la falta de funcionalidades de *CVS*, la comparación entre ambos sistemas se ha convertido en una discusión generalizada entre desarrolladores, incluyendo igualmente manuales y herramientas automatizadas para una fácil y cómoda migración entre sistemas.

Se trata de un sistema de versiones bastante estable y extendido, goza de la reputación de *CVS* como base de su desarrollo y en análisis detallados sobre rendimiento y funcionalidades de cada al desarrollador se presenta como ligero vencedor sobre *CVS*, lo cual ha llevado a progresivas migraciones de los repositorios de compañías de desarrollo software.

- GIT

Este sistema de control de versiones ha crecido enormemente en popularidad en los últimos años [9]. Caracterizado por ser un sistema distribuido, su aplicación inicial sobre desarrollos consistentes en gran cantidad de código fuente ha hecho que se constituya como el modelo de mayor crecimiento del mercado.

Fue ideado e inicialmente desarrollado por uno de los diseñadores de *Linux*, *Linus Torvalds*, quien lanzó su primera versión estable en Abril del año 2005. En la actualidad el desarrollo se encuentra gestionado por *Junio Hamano*, cuya comunidad de desarrolladores ha lanzado la última versión estable 1.9.0 en Febrero del año 2014.

Principalmente este sistema se basa en la creación de nodos que actúan como repositorios locales, los cuales son totalmente independientes del resto, pero que aun así es posible estar sincronizado con el trabajo de los demás desarrolladores [10].

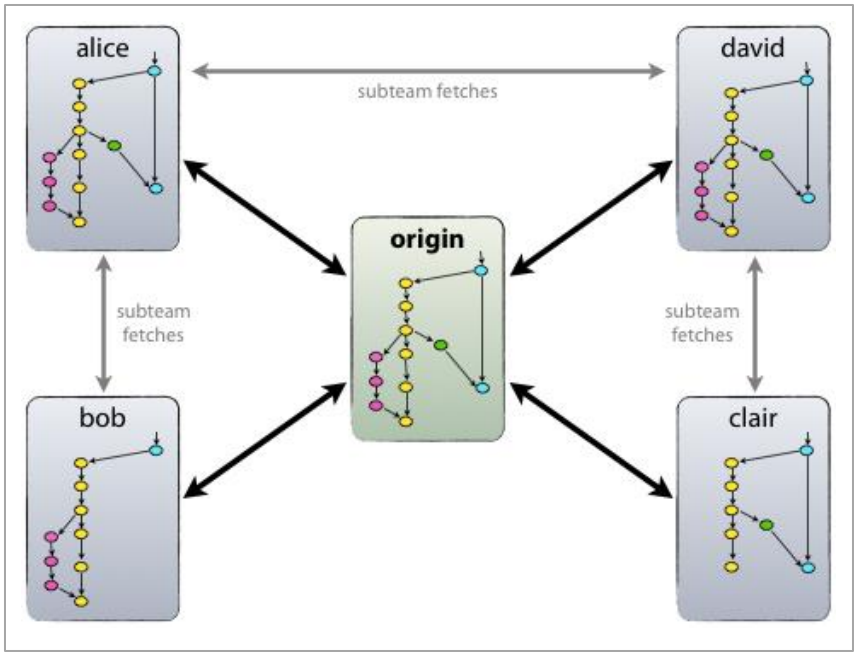


Figura 7. Ejemplo de trabajo distribuido con GIT

La idea de otros sistemas sobre la necesidad de una conexión remota en todo momento es relegada en este modelo por una copia local del repositorio. Con esto, y teniendo en cuenta la experiencia de los desarrolladores y las acciones más comunes que suelen realizar, se llegó a la conclusión de que la gran mayoría de acciones que un desarrollador necesita ejecutar en su día a día pueden ser cubiertas en local sin la necesidad de un acceso remoto.

Otra de las diferencias fundamentales introducida en este modelo es la forma en la que se guardan los cambios en el sistema. Otros modelos como *CVS o Subversion* se basan en almacenar una lista de cambios realizados a partir de un conjunto de ficheros base.

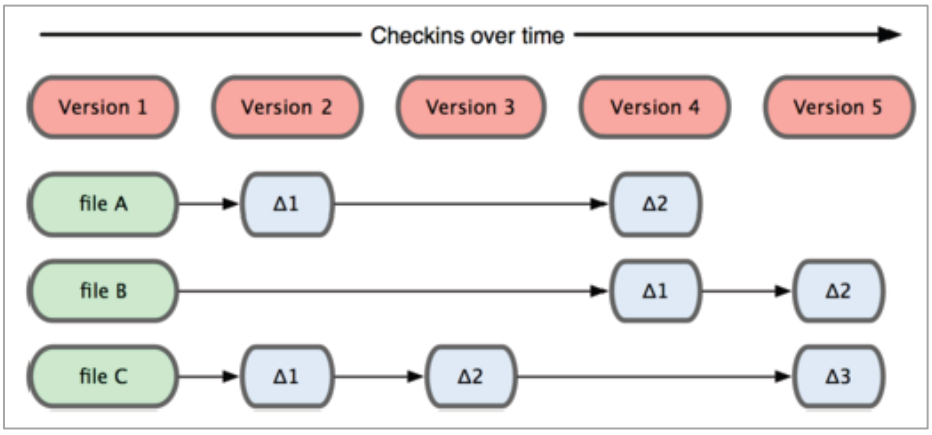


Figura 8. Ejemplo de almacenamiento no GIT

Pero *GIT* considera el versionado como un subconjunto de ficheros, así lo que en cada versión se almacena es el fichero final generado y no una lista de cambios. Hay que remarcar que si un fichero no es modificado, éste no es duplicado, si no que se genera un enlace al fichero original.

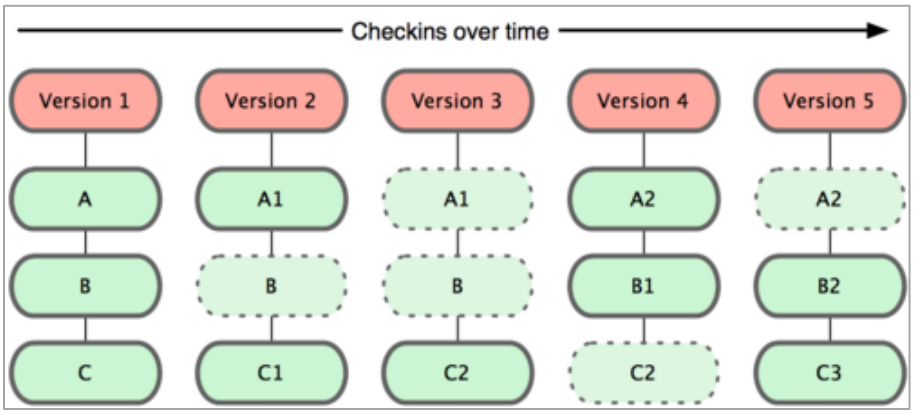


Figura 9. Ejemplo de almacenamiento en GIT

Es importante entender pues las diferentes áreas locales en los que se pueden encontrar los ficheros dentro del modelo de GIT:

- *Working directory*: Es el directorio de trabajo sobre el cual se están realizando los cambios mientras se está desarrollando.

- *Staging Area*: Área en la que se registran los cambios que serán propagados desde el *working directory* al *GIT repository* en el próximo proceso de *commit*. La acción de "*git add*" es la que añadirá estos cambios a la *staging area.*

- *GIT Repository*: Área en la que se encuentran los cambios preparados para ser propagados al repositorio remoto. Los ficheros en este área deben estar listos para ser desplegados en el entorno final.

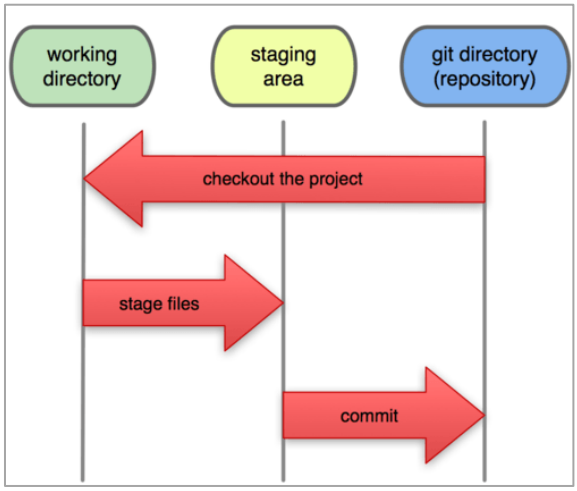


Figura 10. Operaciones en un entorno local GIT

Algunas de las empresas y proyectos más notables que hacen uso de GIT son:

- Kernel de Linux
- Google
- Android
- Microsoft
- Twitter
- LinkedIn
- Eclipse
- Netflix

La gran incursión de este modelo en el mercado y las nuevas funcionalidades introducidas, perfectas para entornos de trabajo distribuidos, hacen de este modelo una opción muy válida para los actuales desarrollos de aplicaciones.

2.3 Diseño final

En los apartados anteriores se han descrito los diferentes componentes que el desarrollo de una aplicación móvil puede necesitar, así como las implementaciones más reseñables para cada uno de los casos.

Es de reseñar que actualmente el desarrollo *open source* está mayormente preparado para poder integrarse con casi todas las posibles soluciones que se puedan encontrar. Así las combinaciones entre las soluciones estudiadas para cada uno de los componentes pueden ser infinitas. Técnicamente es posible integrar cualquier tipo de contenido entre sí.

Son muchos los aspectos a tener en cuenta a la hora de elegir la combinación final, tales como:

- El coste de uso de cada componente, lo que atañe directamente al precio final del desarrollo.
- El nivel de configuración necesario para cada componente, lo que determinará la especialización e incluso el número de recursos necesarios para el proyecto.
- La complejidad en el uso así como las soluciones ofrecidas por cada herramienta, que puede determinar el tiempo necesario para el desarrollo completo.
- Calidad del servicio ofrecido, que definirá directamente la calidad de la propia la aplicación de cara al usuario final.

Estas son algunas de las valoraciones que hay que tener en cuenta a la hora de elegir la solución más óptima con la que desarrollar la aplicación ejemplo de este proyecto.

El estudio pormenorizado de todos los aspectos a definir y de los componentes a utilizar se puede encontrar en el Anexo V. Estudio del Arte: Elección final de componentes.

Las decisiones más relevantes que se han tomado son las siguientes.

Componente	Elección	Detalles
Comunicación cliente - servidor	*Google Cloud Endpoints*	Preparado para dar servicio a las principale plataformas móviles como *Android, HTML5 y iOS*. Además por extensión y compatibilidades de sistemas, compatible con *BlackBerry o Firefox OS.*
Servidor de aplicaciones	*Google App Engine*	*PaaS* que cubre las necesidades específicas del sistema. Compatibilidad con los servicios *Google* aportando valor añadido a la solución final.
Bases de datos	*Object Persistence: JDO*	Compatible y soportado por el servidor de aplicaciones. *JPA* como subconjunto de funcionalidades de *JDO*. Fuertemente integrado con el lenguaje de programación del servidor.
Lenguaje de programación servidor	*JAVA*	Soportado por *Google App Engine* y lenguaje más familiar para el equipo de desarrollo.
Lenguaje de programación cliente	*Android* *HTML5*	Lenguajes propios de los clientes soportados y desarrollados.
Entorno de desarrollo integrado	*Eclipse*	*IDE* preparado para el desarrollo sobre *Google App Engine* y *Android* gracias a plugins específicamente desarrollados.
Sistema de control de versiones	*GITHUB*	De mayor potencial y con mayor cantidad de funcionalidades disponible para un desarrollo distribuido.

Tabla 1. Elección final de componentes

2.4 Conclusiones

En este capítulo dedicado al estudio del arte del mercado actual se han repasado todos los niveles y componentes que pueden formar parte del desarrollo de una aplicación móvil multiplataforma. A partir de ahí se ha elegido el uso del *Cloud Computing* como el mejor modelo para el desarrollo y uso de la lógica de negocio principal, y sobre esa elección se han definido el resto de componentes y modelos a usar para el desarrollo final.

Centrándose en los principales aspectos críticos de un sistema distribuido basado en accesos remotos de clientes sobre un servidor común a través de internet se ha elegido a la tecnología *Google Cloud EndPoints* como la solución más óptima para poner en comunicación las diferentes aplicaciones y servicios que se desarrollarán. Esta tecnología se convierte así en la base del desarrollo de todo el proyecto debido a las soluciones y funcionalidades que ofrece.

Será *Google App Engine* el servidor de aplicaciones que correrá la lógica de negocio principal de la aplicación, desarrollada en lenguaje *JAVA*, y *Eclipse* el *IDE* utilizado para el desarrollo del mismo. El cliente *Android* será desarrollado sobre el entorno de trabajo *Android Studio*. Y como base de datos se elige la especificación *JDO* debido a la gran integración que presenta con el servicio *Google App Engine*.

Todas estas tecnologías, especificaciones y servicios van a ser puestos a prueba mediante un ejemplo explicativo cuyo objetivo es la integración de todos los componentes analizados y mostrar cómo el desarrollo de una aplicación móvil no tiene por qué suponer una batalla entre aplicaciones nativas, aplicaciones *WEB* e incluso aplicaciones híbridas.

El ejemplo elegido se basa en el concepto del *Business Model Canvas* inventado por Alex Osterwalder en el año 2.008 y cuyo canvas puede ser descargado desde su web [11]. Se trata de una estrategia de negocio que puede ser utilizada para la definición de nuevos modelos de negocio aplicados a un producto [12].

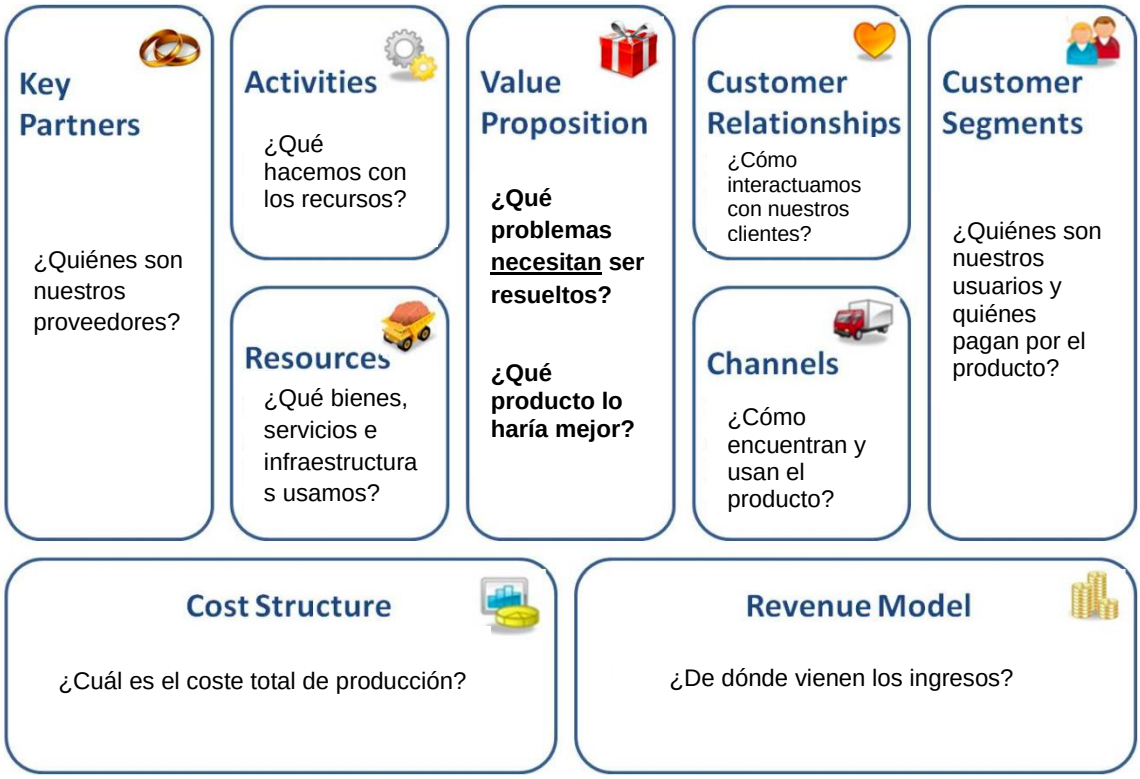


Figura 11. Panel de trabajo para Business Model Canvas

Este modelo se basa en la categorización de los aspectos del modelo de negocio basándose en la naturaleza de los mismos dividiéndolos en 9 áreas principales [13]:

1. Segmentos de clientes. El conjunto de usuarios destinados a consumir el producto.
2. Propuesta de valor. El problema que se solucionará con el producto a ofrecer.
3. Canal. Es el medio con el que se hará llegar la propuesta de valor al segmento de clientes.
4. Relación. Entre los clientes y el servicio prestado.
5. Flujo de ingresos. Importante para definir de antemano los puntos clave del producto.
6. Recursos clave. Aquellos recursos de mayor valor que el servicio necesitará consumir.
7. Actividades clave. Son esas actividades más importantes que el desarrollo del negocio requiere.
8. Alianzas. Relaciones necesarias para que el modelo de negocio se ejecute con garantías.
9. Estructura de costes. Es necesario modelar los costes de la empresa para optimizar los recursos.

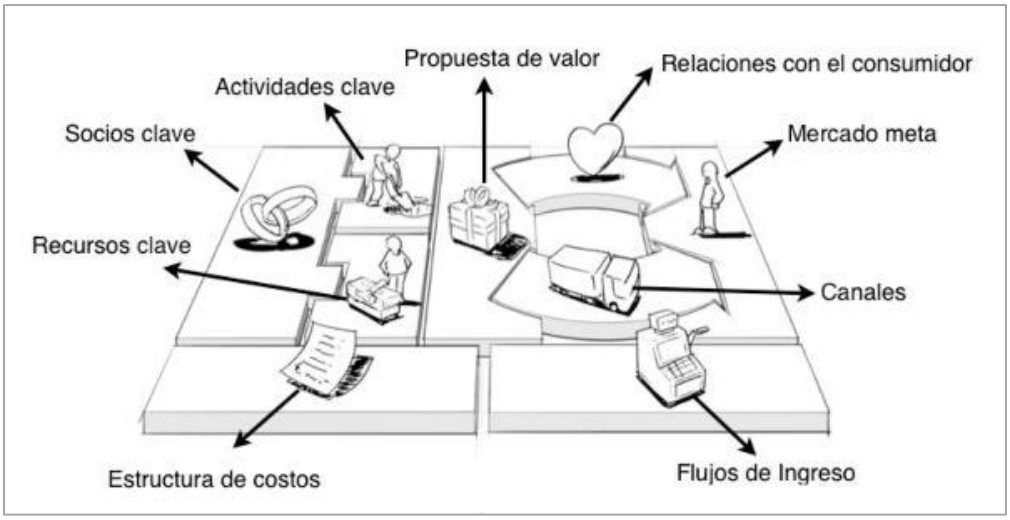


Figura 12. Relaciones entre los componentes de Business Model Canvas

Este modelo se basa en un diagrama donde se representan todos los factores involucrados de un modelo de negocios que pueden ser decisivos para el diseño y crecimiento del mismo.

Durante el diseño del modelo se irán anotando cada una de las ideas aportadas en su sector correspondiente, habitualmente mediante el uso de *post-it* lo que le proporciona mayor dinamismo durante el proceso. El modelo está concebido para fomentar los diseños colaborativos mediante procesos basado en *brainstorming*.

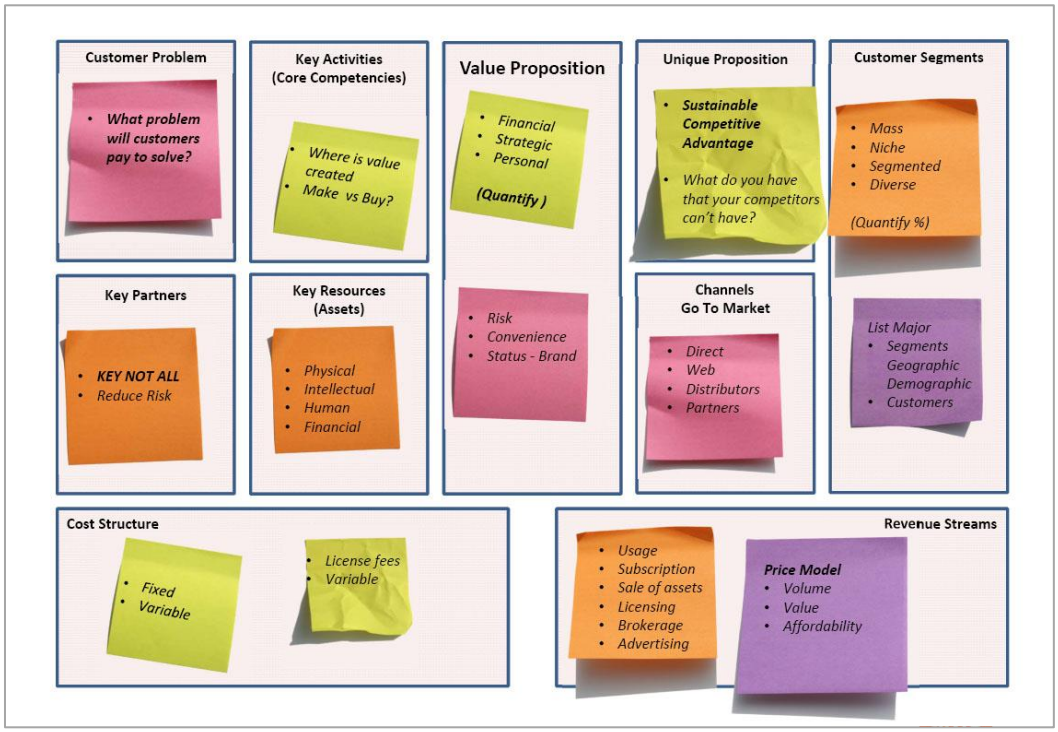


Figura 13. Ejemplo de un panel en el diseño de un modelo de negocio

Capítulo 3

Desarrollo del proyecto

En este capítulo se ha decidido realizar el desarrollo de una aplicación real con el fin de mostrar tanto el proceso completo para el desarrollo de la misma como la aplicación de los componentes finalmente elegidos de entre todos los aspectos estudiados. Se ha elegido para ello el paradigma *Business Model Canvas* para ilustrar las conclusiones de este proyecto de fin de carrera.

3.1 Fase de diseño

El paradigma *Business Model Canvas*, ideado por *Alex Osterwalder*, define una nueva estrategia con la que afrontar el diseño del modelo de negocio de cualquier actividad. Esta estrategia se basa en dividir todos los aspectos de un modelo de negocio en diferentes bloques caracterizados por la actividad que representan. Es importante destacar que estos bloques no son independientes entre sí, sino que en este paradigma se interrelacionan con el fin de representar mejor el valor que cada bloque ofrece y a la vez recoge del resto de actividades.

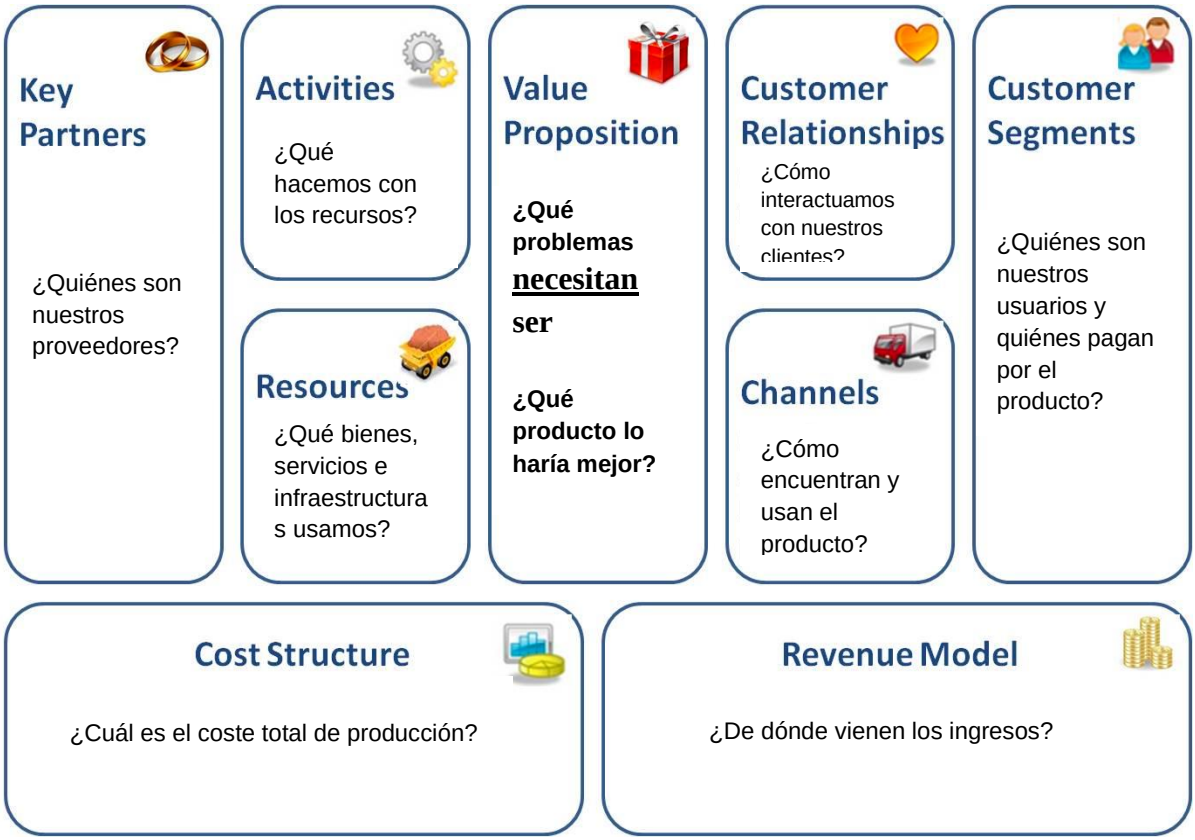


Figura 14. Panel de trabajo para Business Model Canvas

Otro de los aspectos más relevantes de esta estrategia es la del trabajo en equipo de todos los actores involucrados en el diseño del modelo de negocio. Este aspecto encaja perfectamente con el modelo de aplicación que en este proyecto se intenta estudiar, es por esto por lo que se escoge como ejemplo para el desarrollo de una aplicación basada en el modelo propuesto.

3.1.1 Proceso de desarrollo

Para el desarrollo de esta aplicación se decide seguir metodologías *Agile*, específicamente el modelo *Scrum* con el que se hará un seguimiento más detallado de todo el proceso de desarrollo. Se dividirá el proceso en tareas de pequeño tamaño agrupadas en periodos de desarrollo o *Sprints*, con la finalidad de obtener funcionalidades completamente operativas una vez acabado cada *Sprint*.

Para la realización de este proyecto se decide utilizar *Sprint* de 10 horas, periodos fácilmente extrapolables a jornadas laborables estándar de desarrollo de software, con lo que se podrá ver finalmente costes en cuanto a tiempo y recursos que el desarrollo de una aplicación como esta puede requerir.

3.1.2 Requisitos de usuario

A. Basándose en la definición del modelo *Business Model Canvas*, un usuario que interactúe con la aplicación deberá ser capaz de añadir dinámicamente *actividades* al *panel* creado para el modelo de negocio a tratar.

B. Dado que este *panel* será común para todos los *actores* involucrados, las *actividades* creadas deberán ser visibles por el resto de *actores* que hagan uso de la aplicación.

C. La definición de un modelo de negocio es un proceso en constante cambio, por lo que cualquier *actividad* creada puede ser modificada o incluso eliminada en cualquier momento. La modificación de la misma puede ser tanto en contenido (nombre y descripción) como en *actividad* a la que está asociada.

D. Los usuarios que harán uso de la aplicación lo podrán hacer desde el mayor número de plataformas y dispositivos posibles.

3.2 Fase de desarrollo

Teniendo en cuenta las definiciones anteriores tanto del modelo a implementar como de los requisitos de usuario definidos el desarrollo de la aplicación tendrá las siguientes definiciones.

3.2.1 Especificación de requisitos

Se listan los requisitos fundamentales del que se compondrá el sistema.

A. Se utilizará un servidor *Google App Engine* que correrá la lógica de negocio relacionada con la gestión de las *actividades* creadas en la herramienta, tales como la creación, modificación y eliminación de las mismas.

B. Gracias a la tecnología *Google Cloud Endpoints* se generarán las librerías necesarias por los clientes para hacer uso de esta lógica de negocio.

C. Se deben desarrollar la mayor cantidad de aplicaciones cliente posible a partir de las librerías proporcionadas: *HTML, Android e iOS*.

D. Haciendo uso del concepto de *responsive design* se deberá además proporcionar solución a las diferentes plataformas que hagan uso de las aplicaciones. *HTML* tanto para *WEB* como para móvil, *Android* tanto para dispositivos móviles como para *tablets*, etc.

E. Dado que el objetivo principal de este proyecto es proporcionar lógica de negocio centralizada que proporcione servicio distribuido a diferentes clientes, se desarrollará una herramienta adicional para la representación del uso de esta lógica en tiempo real.

3.2.2 Modelos de Bases de Datos

La entidad más importante de este proyecto son las *actividades* que contendrá el panel que representa el *Business Model Canvas*.

Dado que la tecnología de bases de datos que se usará en este ejemplo es *JDO*, el diseño de la base de datos es en sí el diseño de la entidad *JAVA* que representará a esas actividades. *JDO* se encargará automáticamente de la traslación de esta entidad *JAVA* al modelo de datos más idóneo dependiendo de la naturaleza del mismo. En este caso, la entidad es la siguiente:

CanvasItem	
id:	Long
category:	String
title:	String
description:	String
author:	String

- *id*: Identificador único de cada *actividad*. Autogenerado en el momento de la creación y almacenamiento de la misma en base de datos. Usado para su identificación a la hora de modificar y eliminar las actividades.

- *category*: Representa cada una de los bloques en los que el modelo de negocio puede ser dividido y al que cada *actividad* puede estar asignado.

- *title*: Título de la *actividad*. Texto representativo que será mostrado en todo momento. Para plataformas con visualización menor será el único texto a ser mostrado.

- *description*: Texto descriptivo de la *actividad*. Dará información más detallada sobre la *actividad* que se ha creado. Para plataformas con visualización mayor se mostrará este texto acompañando al título principal de la *actividad*.

- *author*: Representa al actor que ha creado la *actividad*.

La implementación de esta representación de datos se hace sobre la clase *CanvasItem.java* que es anotada apropiadamente según las especificaciones de la tecnología *JDO*.

```
@PersistenceCapable
public class CanvasItem {

	@PrimaryKey
	@Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
	private Long id;

	@Persistent
	private String category;

	@Persistent
	private String title;

	@Persistent
	private String description;

	@Persistent
	private String author;
}
```

Código 1. Definición del modelo de base para el ejemplo Business Model Canvas

La clase debe ser anotada como *@PersistenceCapable* para que *JDO* la interprete como un objeto que debe ser almacenado en bases de datos. Cada uno de los parámetros que se desean almacenar deben ser anotados igualmente como *@Persistent*.

El parámetro *id* debe ser anotado además como *@PrimayKey* que identifica a este parámetro como la clave primaria de la tabla. Adicionalmente se decide que este valor sea automáticamente generado en el momento en que cada objeto es insertado en la base de datos, por ello la anotación *@Persistent* debe ser configurada como *valueStrategy = IdGeneratorStrategy.IDENTIFY*.

Con esta configuración la representación en bases de datos de estas entidades es la siguiente:

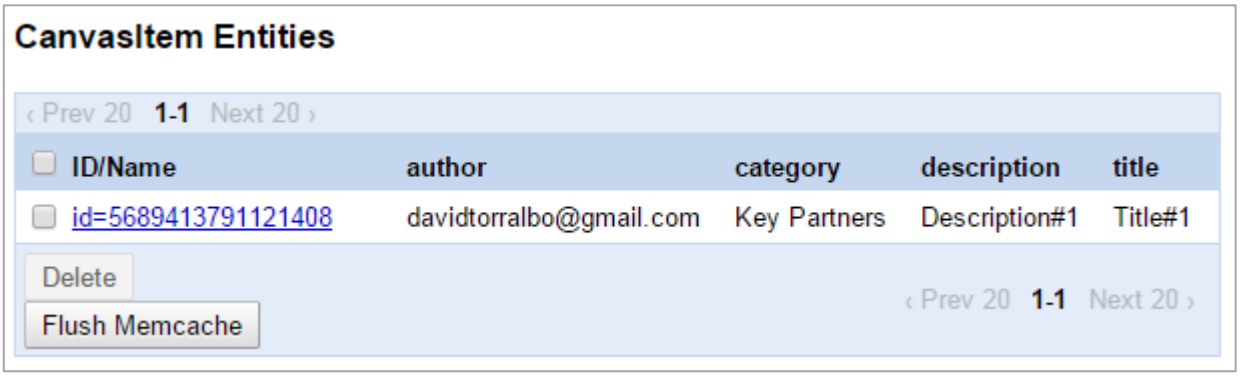


Figura 15. Representación de un objeto CanvasItem en base de datos en App Engine

3.2.3 Catálogo de Servicios

Una vez se ha creado y anotado adecuadamente el modelo de datos es posible crear la base de la lógica de negocio del servicio gracias a la tecnología *Google Cloud EndPoints*. Basta con ejecutar la acción "*Generate Cloud Endpoint Class*" sobre la entidad anterior y se obtendrán los servicios básicos necesarios.

Google Cloud EndPoints utiliza el modelo de datos para generar una *API* que será consumida por clientes externos en base a los servicios configurados en la clase *CanvasItemEndpoint.java*.

Sobre la *API* generada automáticamente es posible aplicar algunas modificaciones que harán los servicios más legibles y reconocibles, tanto para el uso y pruebas internas como para el consumo por parte de los clientes finales.

- Descripción de la API

La configuración por defecto creada siguiente puede ser modificada para simplificar su estructura y generar una *API* más legible, por ejemplo en cuanto al nombre se refiere.

```
@Api(name = "canvasitemendpoint",
     namespace = @ApiNamespace(ownerDomain = "dtorralbo.com",
                               ownerName = "dtorralbo.com",
                               packagePath = "bmca"))
public class CanvasItemEndpoint {...}
```

Código 2. Configuración por defecto de la API

Es posible modificar el nombre para hacer uso de uno más representativo de la aplicación y añadir una nueva configuración que ayude al control de versiones, lo cual evitará problemas de uso de versiones cuando el servicio esté disponible a los clientes. Así el resultado sería el siguiente.

```
@Api(name = "bmca",
     version = "v1")
public class CanvasItemEndpoint {
    [...]
}
```

Código 3. Reconfiguración de la API

- Descripción de los servicios estándar

Google Cloud EndPoints genera los servicios básicos *CRUD (Create, Read, Update and Delete)* que una *API* necesita. Y al igual que en la descripción de la *API*, esta tecnología permite modificar la descripción de estos servicios con el objetivo de presentarlos al cliente de una forma más simple y entendible.

```
@SuppressWarnings({ "unchecked", "unused" })
@ApiMethod(name = "listCanvasItem")
public CollectionResponse<CanvasItem> listCanvasItem(
    @Nullable @Named("cursor") String cursorString,
    @Nullable @Named("limit") Integer limit) {...}

@ApiMethod(name = "getCanvasItem")
public CanvasItem getCanvasItem(@Named("id") Long id) {...}

@ApiMethod(name = "insertCanvasItem")
public CanvasItem insertCanvasItem(CanvasItem canvasitem) {...}

@ApiMethod(name = "updateCanvasItem")
public CanvasItem updateCanvasItem(CanvasItem canvasitem) {...}

@ApiMethod(name = "removeCanvasItem")
public void removeCanvasItem(@Named("id") Long id) {...}
```

Código 4. Descripción de los servicios básicos generados

Así, estos servicios creados de forma automática pueden ser presentados de la siguiente forma.

```
@SuppressWarnings({ "unchecked", "unused" })
@ApiMethod(name = "Item.list")
public CollectionResponse<CanvasItem> listCanvasItem(
    @Nullable @Named("cursor") String cursorString,
    @Nullable @Named("limit") Integer limit) {...}

@ApiMethod(name = "Item.get")
public CanvasItem getCanvasItem(@Named("id") Long id) {...}

@ApiMethod(name = "Item.add")
public CanvasItem insertCanvasItem(CanvasItem canvasitem) {...}

@ApiMethod(name = "Item.update")
public CanvasItem updateCanvasItem(CanvasItem canvasitem) {...}

@ApiMethod(name = "Item.delete")
public void removeCanvasItem(@Named("id") Long id) {...}
```

Código 5. Reconfiguración de los servicios básicos

- Creación de nuevos servicios

Además la tecnología *Google Cloud EndPoints* ofrece la posibilidad de añadir nuevos servicios que puedan ser necesarios y que no sean cubiertos por los servicios estándar generados originalmente. Para ellos solo es necesario añadir dicha funcionalidad y anotarla acorde a los servicios anteriores.

En este caso se ha considerado oportuno proporcionar un servicio que actualice la categoría a la que una actividad asociada, correspondiente a la acción de cambiar una actividad de una categoría a otra.

Esta funcionalidad puede ser cubierta mediante el servicio ya presente *Item.update*, pero para ello es necesario generar el objeto *CanvasItem* completo por parte del cliente para consumir el servicio. Para optimizar la ejecución únicamente será necesario indicar el *id* de la actividad a modificar así como la nueva categoría.

El nuevo servicio creado tendría la siguiente configuración.

```
@ApiMethod(name = "Item.changeCategory",
           httpMethod = HttpMethod.POST)
public CanvasItem changeCanvasItemCategory(@Named("id") Long id,
    @Named("category") String newCategory) {
    [...]
}
```

Código 6. Creación de nuevos servicios

- Lista final de servicios disponibles

Tras la configuración de servicios, el catálogo que será ofrecido a los clientes es el siguiente.

Nombre del Servicio (v1)	Parámetros	Descripción
bmca.item.add	CanvasItem canvasItem	Añadir una nueva actividad
bmca.item.delete	Long id	Eliminar la actividad correspondiente al id
bmca.item.update	CanvasItem canvasItem	Actualizar una actividad
bmca.item.get	Long id	Recuperar la actividad correspondiente al id
bmca.item.list		Listar todas las actividades disponibles
bmca.item.changeCategory	Long id, String newCategory	Actualizar la categoría a la actividad indicada

Tabla 2. Servicios ofrecidos en BMCA

3.2.4 Publicación de los servicios

Una vez que los servicios han sido creados y configurados acorde a las necesidades del proyecto es posible publicarlos para ser consumidos por los clientes finales.

Antes de este paso, se recomienda probar todos los servicios que serán publicados en el entorno local que el *plugin* de *Eclipse* proporciona. Para ello se puede utilizar la herramienta *API Explorer* que puede ser accedida en la siguiente dirección:

- *http://localhost:8888/_ah/api/explorer*

Este mismo recurso está disponible cuando los servicios son publicados y es de gran ayuda para el desarrollo de la lógica de negocio ya que pueden ser completamente probados antes de entregar las librerías cliente a los desarrolladores de las aplicaciones cliente. Gracias a esto es posible reducir los tiempos de integración entre componentes (cliente y servidor) ya que con esta herramienta es posible simular fielmente las peticiones y llamadas que los clientes realizarán sobre la lógica de negocio presentada.

Para la publicación final de los servicios es necesario crear un nuevo proyecto dentro de *App Engine* usando para ellos la consola presente en la siguiente dirección:

- *https://appengine.google.com/*

Es necesario elegir un identificador único que representará a la aplicación servidora y que será constituirá la *URL* por defecto de los servicios.

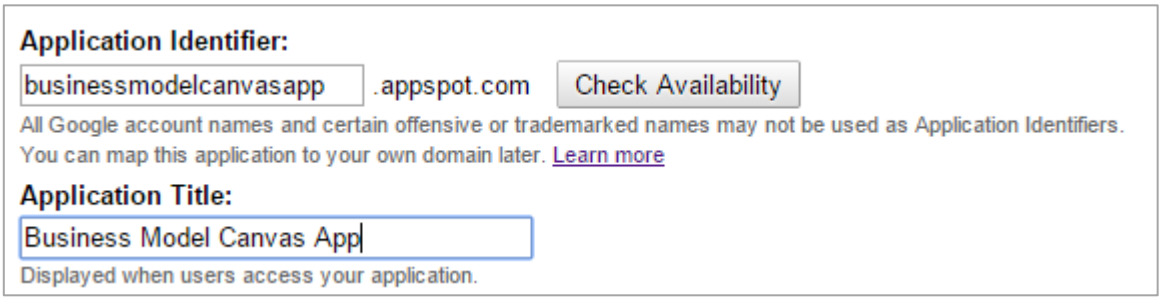


Figura 16. Creación de una aplicación en la consola de App Engine

Una vez configurados estos parámetros solamente es necesario configurar el descriptor de nuestra lógica de negocio para poder publicar los servicios. Para ello se debe especificar el identificador elegido dentro del fichero de configuración "*appengine-web.xml*".

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>businessmodelcanvasapp</application>
  <version>1</version>

  [...]

</appengine-web-app>
```

Código 7. Configuración del descriptor de App Engine

Con esta configuración ya es posible publicar los servicios haciendo uso de la funcionalidad que el *plugin* de *Eclipse* proporciona.

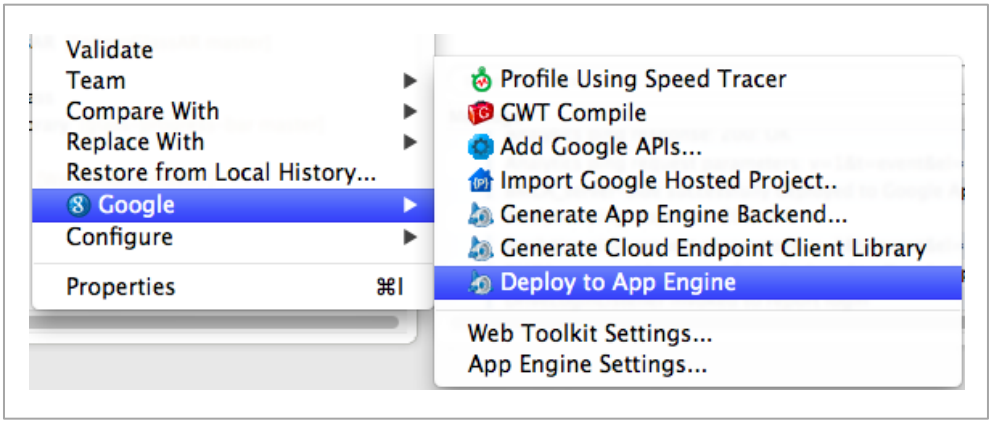


Figura 17. Despliegue de una aplicación en el servidor de App Engine desde Eclipse

Los servicios ya públicos pueden ser probados por diferentes herramientas (curl, RESTClient, etc) pero en este caso *App Engine* proporciona la herramienta *API Explorer* que simplifica enormemente este proceso. En este caso se puede acceder a la herramienta localizada en la siguiente dirección:

- *http://businessmodelcanvasapp.appspot.com/_ah/api/explorer*

Desde esta herramienta es posible probar los distintos servicios que se encuentran disponibles.

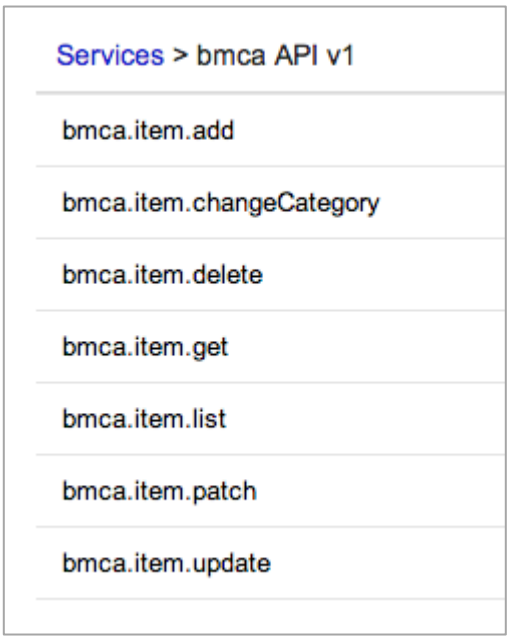


Figura 18. Listado de servicios disponible para un servicio en la herramienta API Explorer de Google

Así por ejemplo se podría añadir una nueva actividad o ítem.

Figura 19. Ejemplo de datos necesarios para añadir un nuevo item

En la herramienta es posible ver la petición final que se realiza así como la respuesta obtenida.

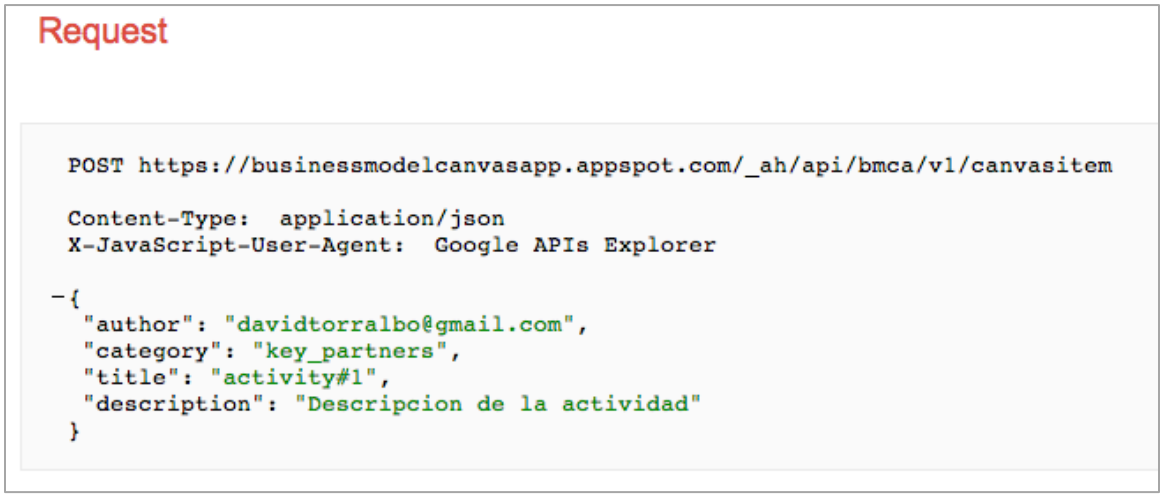


Figura 20. Petición realizada desde la herramienta API Explorer

Figura 21. Respuesta a la petición realizada en la Figura 20

3.2.5 Generación de las librerías cliente

Tras probar todos los servicios se deben generar las librerías que serán consumidas por los clientes mediante el *plugin* para *Eclipse*.

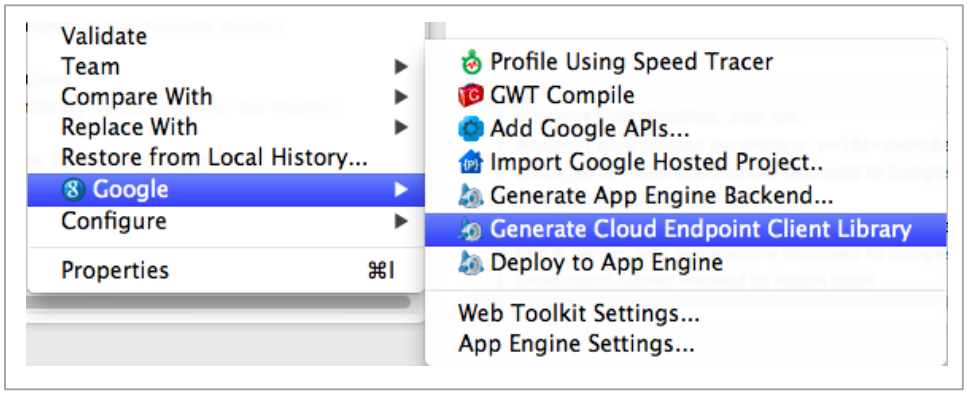


Figura 22. Generación de los Endpoints mediante el plugin de Eclipse

Esta acción genera una nueva estructura correspondiente a las librerías que se usarán para poder acceder a los servicios creados.

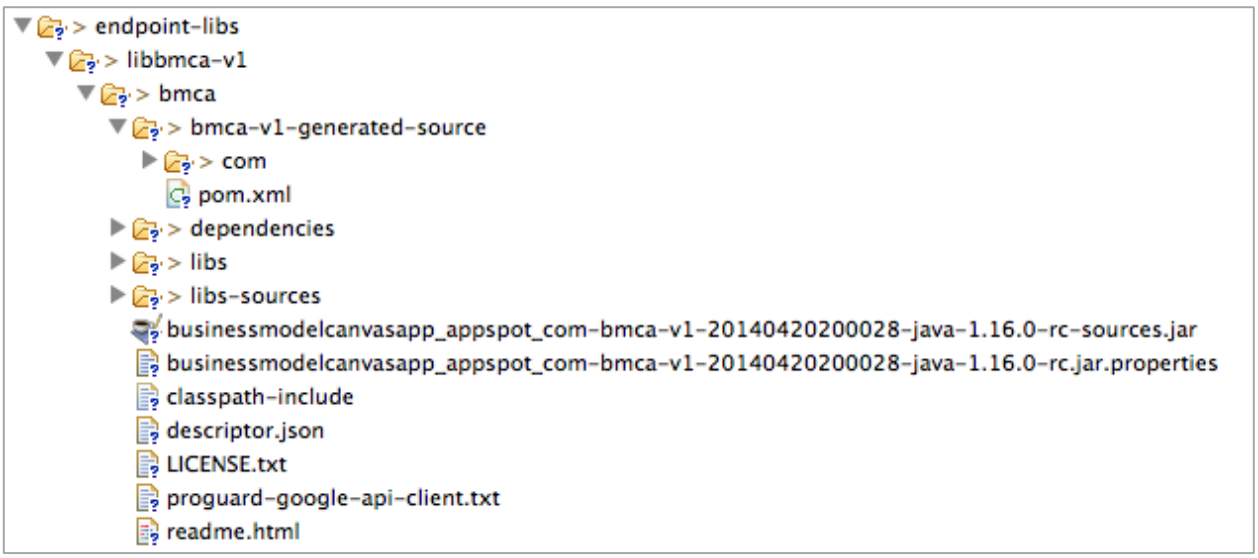


Figura 23. Estructura de clases generada por el plugin de Eclipse

Para el caso del cliente *Android*, existen diversas formas de hacer uso de estos *endPoints* generados, ya que al crearlos se proporciona el código fuente en la librería:

- *businessmodelcanvasapp_appspot_com-bmca-v1-20140420200028-java-1.16.0-rc-sources.jar*

Simplemente habría que descomprimir este *jar* dentro del proyecto *Android* cliente para poder acceder a estos servicios. Pero dado que es posible que sean clientes externos quienes consuman estos servicios es más recomendable proporcionar una librería finalmente compilada. Para ello se puede hacer uso del proyecto *maven* que se ha generado bajo el directorio "*bmca-v1-generated-source*".

Hay que prestar atención a la configuración por defecto que aparece en el fichero *pom.xml*:

```
<groupId>com.google.apis</groupId>
<artifactId>google-api-services-bmca</artifactId>
<version>v1-rev20140420200028-1.16.0-rc</version>
<name>bmca v1 (revision 20140420200028)</name>
<packaging>jar</packaging>
```

Código 8. Configuración por defecto del fichero pom.xml generado automáticamente

Esta configuración puede ser mejorada para obtener una *API* más legible:

```
<groupId>com.google.apis</groupId>
<artifactId>bmca-client</artifactId>
<version>1.0.0.0-SNAPSHOT</version>
<name>bmca v1</name>
<packaging>jar</packaging>
```

Código 9. Reconfiguración del fichero pom.xml

Así ya se puede compilar el proyecto mediante el comando *package* de *maven*..

```
$ mvn clean package
[INFO] ------------------------------------------------------------------------
[INFO] Building bmca v1 1.0.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------

       [...]

[INFO] --- maven-jar-plugin:2.3.1:jar (default-jar) @ bmca-client ---
[INFO] Building jar: /Development/git/BusinessModelCanvasApp/server-side/endpoint-
libs/libbmca-v1/bmca/bmca-v1-generated-source/target/bmca-client-1.0.0.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.017 s
[INFO] Final Memory: 11M/81M
[INFO] ------------------------------------------------------------------------
```

Código 10. Ejecución del comando package sobre los endpoints creados

Con esto se consigue obtener la librería ***bmca-client-1.0.0.0-SNAPSHOT.jar*** que será utilizada por los clientes *Android*, localizada bajo el directorio *target* del dicho proyecto *maven*.

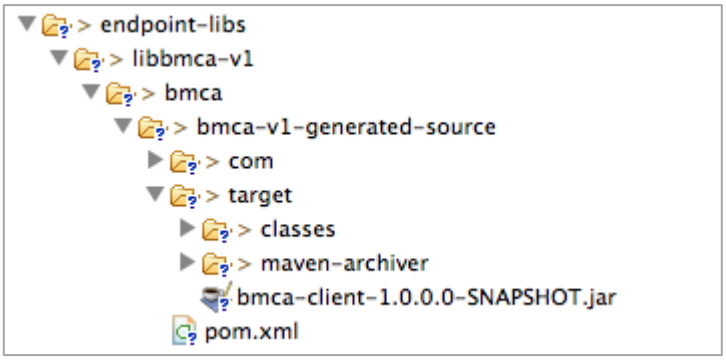


Figura 24. Librería cliente final generada mediante maven

3.2.6 Desarrollo del cliente Android

Para ello se crea un nuevo proyecto *Android* en *Eclipse* que generará la estructura básica de una aplicación *Android*.

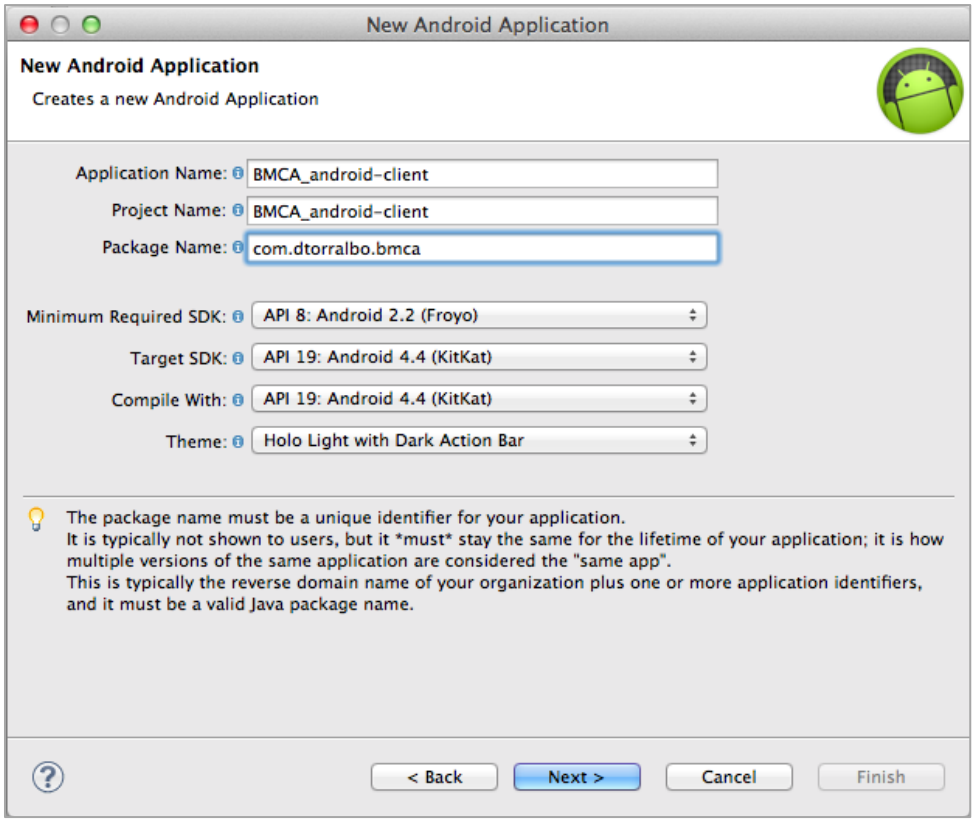


Figura 25. Página de creación de una aplicación Android en Eclipse

Para este proyecto se desarrolla la aplicación orientada a dispositivos móviles de pantallas reducidas, por lo que la visualización completa del panel de *Business Model Canvas* no es posible. Para solucionar esto se decide usar la visualización de las categorías usando el diseño de *Swipe Views*, con el que cada pantalla corresponde a una categoría del panel.

Figura 26. Maquetación de las pantallas de la aplicación Android basada en Swipe Views

La aplicación proporciona las funcionalidades requeridas de listar, crear y actualizar las actividades presentes en el panel.

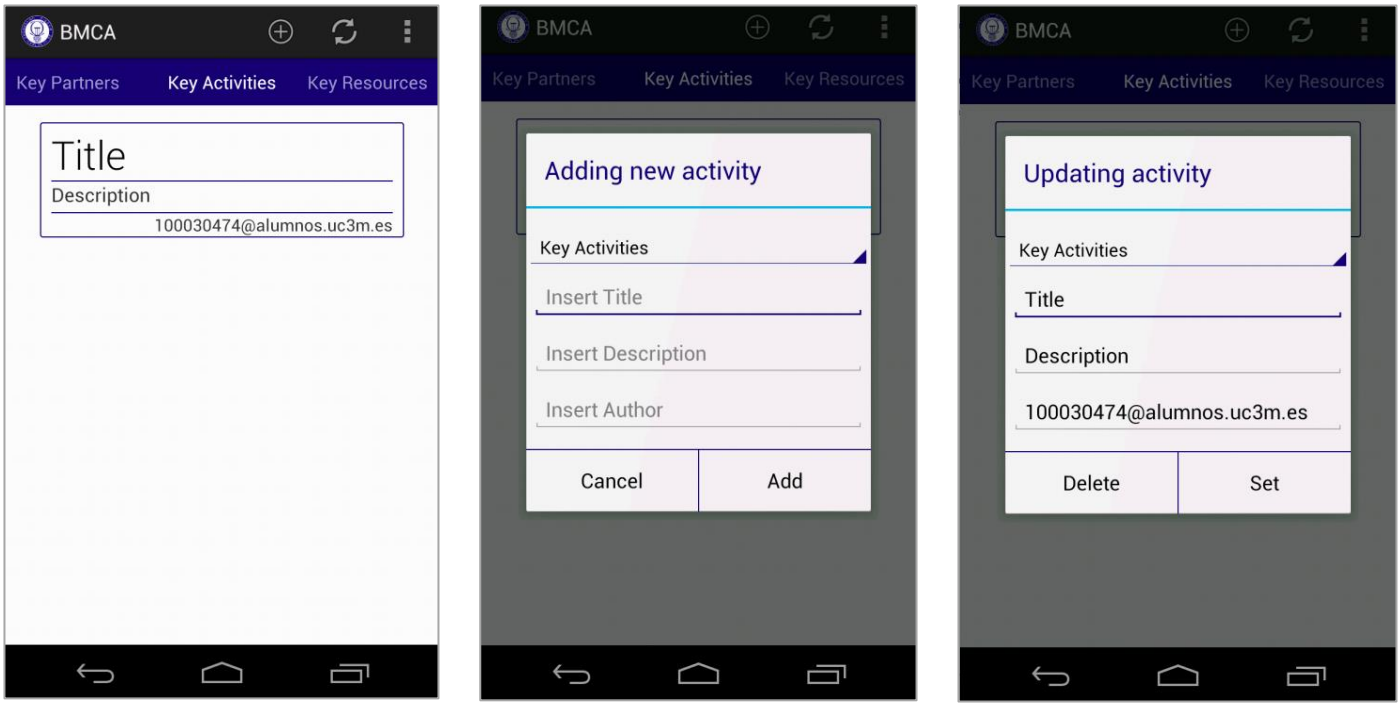


Figura 27. Listar, crear y modificar actividades

En este proyecto se ha decidido centralizar la lógica para el acceso a los datos de la aplicación en una única clase *CanvasItemsManager.java*, dejando el desarrollo de la interfaz independiente del modelo de datos. Esto ha ayudado a un desarrollo desacoplado que mejora en tiempos y en rendimiento del equipo.

Una vez la interfaz está acaba, el acceso a los datos de la aplicación se hará utilizando la librería cliente que generada en *back-end (bmca-client-1.0.0.0-SNAPSHOT.jar)*.

Para un uso óptimo no solo son necesarias las librerías cliente generadas, sino librerías que encapsulan el acceso a los recursos mediante el acceso a internet. Estas librerías son generadas por el *plugin* de *Eclipse* en el momento de generar los *endpoints* y deben ser importadas como dependencias del proyecto *Android*.

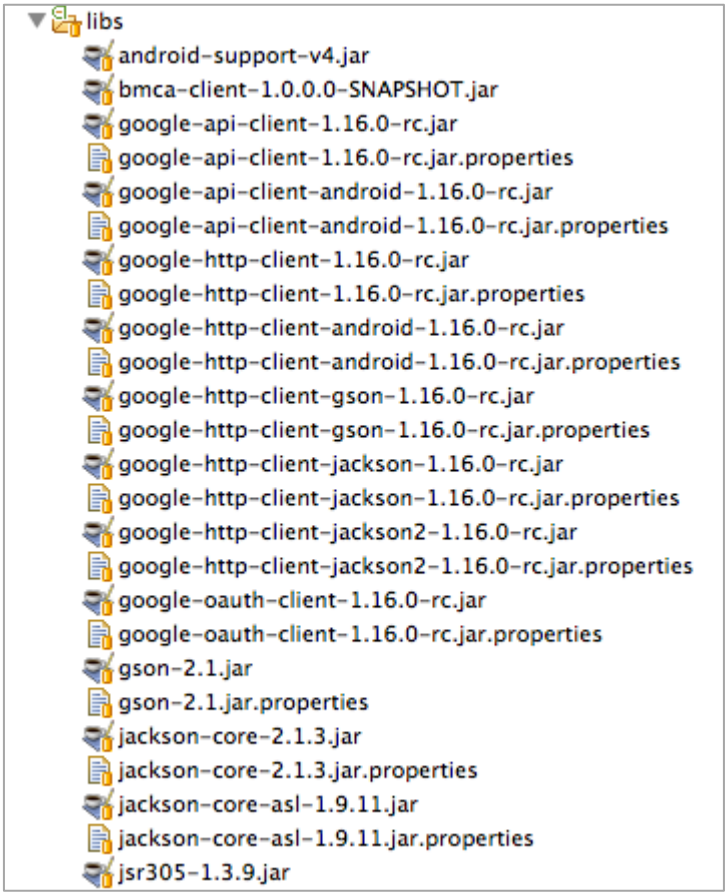


Figura 28. Librerías necesarias para el uso del cliente generado

Además, dado que se hará uso de la conexión a internet para acceder a los servicios y a sus datos, hay que estar seguros de configurar los permisos necesarios. Para ello hay que añadir en el descriptor del proyecto *Android* (*AndroidManifest.xml*) el siguiente permiso:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Código 11. Configuración de los permisos para acceso a Internet en Android

Como se recomienda en el desarrollo de aplicaciones *Android*, el accesos a recursos que pueden conllevar acceso a internet o a bases de datos, o en general acciones que puedan conllevar un bloqueo temporal en el hilo principal de la aplicación, deben ser realizadas en un hilo secundario con el fin de no impactar a la experiencia del usuario ni bloquear la aplicación en sí.

El recurso más común en *Android* es el uso de tareas asíncronas (*AsyncTask*). Este recurso es utilizado tanto si se accede a recursos en la nube como si se accede a bases de datos locales en el terminal. El uso de estas tareas por tanto habría sido el mismo en cualquier caso.

Con la uso de una tarea asíncrona se consigue la ejecución de la lógica en un hilo paralelo, así el usuario no quedará bloqueado. Una vez la tarea haya concluido, es posible notificar al hilo principal para que actúe según convenga, ya sea modificando la interfaz del usuario o incluso ejecutando otras funcionalidades de la aplicación.

En este caso se ha decidido utilizar *listener* propios para obtener la notificación de tarea terminada así como para obtener el resultado de la misma, generalmente datos recuperados de los servicios en la nube.

Una vez que las tareas han sido definidas y enlazadas mediante los *listener* con el interfaz principal de la aplicación, el desarrollo se centra en el consumo de los servicios de la *API* mediante la librería *endpoint* generada. El desarrollo propio para estas funciones son:

- Inicialización de la *API* que conectará con el *back-end*, necesario en todo momento para poder ejecutar cualquiera de los servicios.

```
Bmca.Builder builder = new Bmca.Builder(AndroidHttp.newCompatibleTransport(),
                                        new JacksonFactory(),
                                        null);
Bmca service = builder.build();
```

Código 12. Creación del objeto Bmca sobre el que ejecutar los servicios

- Listar las actividades existentes.

```
CollectionResponseCanvasItem itemsCollection = service.item().list().execute();

List<CanvasItem> items = itemsCollection.getItems();
```

Código 13. Servicio para listar todas las actividades desde Android

- Añadir una nueva actividad.

```
service.item().add(item).execute();
```

Código 14. Añadir una nueva actividad desde la app Android

- Modificar una actividad existente.

```
service.item().update(item).execute();
```

Código 15. Servicio para actualizar una actividad

- Eliminar una actividad.

```
service.item().delete(id).execute();
```

Código 16. Eliminar una actividad mediante Google Cloud Endpoints

En este caso el servicio principal es un objeto de tipo “*Bmca*”, que es el nombre con el que se determinó a la *API* en *back-end*.

```
@Api(name = "bmca",
     version = "v1")
public class CanvasItemEndpoint {
      [...]
}

Bmca service = builder.build();
```

Código 17. Relación entre la configuración de la API en el servidor y el objeto en el cliente Android

Y cada uno de los métodos utilizados corresponde a los servicios generados en la *API*.

- Listar las actividades existentes.

```
@SuppressWarnings({ "unchecked", "unused" })
@ApiMethod(name = "Item.list")
public CollectionResponse<CanvasItem> listCanvasItem(
    @Nullable @Named("cursor") String cursorString,
    @Nullable @Named("limit") Integer limit) {

    [...]

}

service.item().list().execute();
```

Código 18. Relación entre el servicio Item.list en el servidor y el método item().list() en el cliente

- Añadir una nueva actividad.

```
@ApiMethod(name = "Item.add")
public CanvasItem insertCanvasItem(CanvasItem canvasitem) {

    [...]

}

service.item().add(item).execute();
```

Código 19. Relación entre el servicio Item.add en el servidor y el método item().add() en el cliente

- Modificar una actividad existente.

```
@ApiMethod(name = "Item.update")
public CanvasItem updateCanvasItem(CanvasItem canvasitem) {

    [...]

}

service.item().update(item).execute();
```

Código 20. Relación entre el servicio Item.update en el servidor y el método item().update() en el cliente

- Eliminar una actividad.

```
@ApiMethod(name = "Item.delete")
public void removeCanvasItem(@Named("id") Long id) {

    [...]

}

service.item().delete(id).execute();
```

Código 21, Relación entre el servicio Item.delete en el servidor y el método item().delete() en el cliente

Es posible ver con este ejemplo la simplicidad en el desarrollo para consumir los servicios disponibles por la *API* y alojada en la nube. Solo un simple método es necesario para poder ejecutar la lógica en el servidor, esto simplifica notablemente la tarea del desarrollador ya que los EndPoints encapsulan toda la lógica necesaria para hacer la petición, sin necesidad de atender al establecimiento de la conexión entre las dos entidades, la codificación y decodificación del mensaje enviado, etc.

3.2.7 Desarrollo del cliente WEB

Este cliente, como el cliente *Android*, servirá de herramienta de comunicación entre el servidor central y los clientes que hagan uso de la plataforma *WEB* como aplicación. Este cliente puede ser accedido desde la siguiente dirección.

- http://businessmodelcanvasclient.appspot.com

Para el desarrollo de este cliente se decide usar las librerías *JQuery* y *JQuery Mobile*. Los *widgets* que proporciona para desarrollar sobre *HTML5* hacen que la implementación sea más sencilla y mejor adaptada a aplicaciones *WEB Mobile*. Para ello es necesario incluir las librerías correspondientes.

```
<script type='text/javascript' src='http://code.jquery.com/jquery-1.7.2.js'/>

<script type="text/javascript"
src='http://code.jquery.com/mobile/1.3.0/jquery.mobile-1.3.0.js'/>

<link type="text/css" rel="stylesheet"
href="http://code.jquery.com/mobile/1.3.0/jquery.mobile-1.3.0.css"/>
```

Código 22. Carga de las librerías JQuery utilizadas en el ejemplo

Siguiendo el diseño del cliente *Android*, se elige el modelo de una página por categoría para crear la aplicación. En cada página se listan las actividades asociadas a cada categoría con la posibilidad de añadir nuevas actividades y de modificar o incluso eliminar cualquiera de las ya existentes.

Para ello, cada una de las páginas contará con una cabecera que alojará las funciones de añadir una nueva actividad así como la de refrescar el contenido actual. El cuerpo en sí de cada página listará, de forma dinámica, las actividades correspondientes.

```
<div data-role="page" id="Key Partners">
	<div data-role="header" data-position="fixed">
		<div id="headerGroup">
			<div data-role="controlgroup" data-type="horizontal" >
				<a href="#" data-role="button"
onclick="showNewItemPage();" style="padding: 15px;">Add</a>
				<a href="#" data-role="button" onclick="refreshItems();"
style="padding: 15px;">Refresh</a>
			</div>
		</div>

		<div class="category">Key Partners</div>
	</div>
	<div data-role="content">
	</div>
</div>
```

Código 23. Creación de cada página en el cliente WEB

Adicionalmente se crean las páginas que mostrarán el formulario para crear, modificar y eliminar actividades.

```
<div data-role="page" id="Update Item" data-close-btn="none">
	<div data-role="header">
		<div class="category">Update Activity</div>
	</div>

	<div data-role="content">
		<div data-role="fieldcontain" id="updateCategory" style="margin:0 auto; margin-
left:auto; margin-right:auto; align:center; text-align:center;">
			<select id="selectCategory">
				<option value="Key Partners">Key Partners</option>
				<option value="Key Activities">Key Activities</option>
				<option value="Key Resources">Key Resources</option>
				<option value="Value Propositions">Value Propositions</option>
				<option value="Customer Relationships">Customer Relationships</option>
				<option value="Channels">Channels</option>
				<option value="Customer Segments">Customer Segments</option>
				<option value="Cost Structure">Cost Structure</option>
				<option value="Revenue Streams">Revenue Streams</option>
			</select>
		</div>
		<input type="hidden" id="updateId"/>
		<input type="text" id="updateTitle" placeholder="Title" />
		<input type="text" id="updateDescription" placeholder="Description" />
```

```
            <input type="text" id="updateAuthor" placeholder="Author" />
            <div id="navgroup">
                  <div data-role="controlgroup" data-type="horizontal" style="margin-top: 100px">
                        <a href="#" data-role="button" id="cancel" style="color: #000e77;"
onclick="backPage();">Cancel</a>
                        <a href="#" data-role="button" id="delete" style="color: #c42d2d;"
onclick="google.endpoints.bmcaApi.remove();">Delete</a>
                        <a href="#" data-role="button" id="update" style="color: #000e77;"
onclick="google.endpoints.bmcaApi.update();">Update</a>
                  </div>
            </div>
      </div>
</div>
```

Código 24. Implementación de la página para actualizar una actividad del panel

Una vez se han creado las páginas es necesario enlazar la aplicación *WEB* con los servicios disponibles en el servidor. Para ello se debe cargar el cliente *javascript* proporcionado por *Google* que facilita esta comunicación, y una vez cargado es posible inicializar la comunicación con los *endPoints*.

```
<script type="text/javascript">
      initialize = function() {
            registerListener();

            google.endpoints.bmcaApi.init(
                  'https://businessmodelcanvasapp.appspot.com/_ah/api'
            );
      }
</script>
<script src="https://apis.google.com/js/client.js?onload=initialize"></script>
```

Código 25. Código para cargar los Endpoints del servidor desde el cliente WEB

Viéndolo esto en detalle, se realiza la carga de la librería *javascript* y se configura que, cuando se haya terminado de cargar, se ejecute la función '*initialize*'.

```
<script src="https://apis.google.com/js/client.js?onload=initialize"></script>
```

Código 26. Script para cargar la librería de Google para poder hacer uso de los Endpoints

Y dentro de esta función, además de registrar los *listener* que manejan la navegación entre páginas, se inicializa la comunicación con los *endPoints* que el servidor proporciona.

```
google.endpoints.bmcaApi.init('https://businessmodelcanvasapp.appspot.com/_ah/api');
```

Código 27. Inicialización de los Endpoints alojados en la dirección especificada

Esta función '*init*' es la encargada de crear la comunicación y de, adicionalmente, recuperar todas las actividades existentes para ser listadas en la aplicación.

```
google.endpoints.bmcaApi.init = function(apiRoot) {
	var callback = function() {
		google.endpoints.bmcaApi.list();
	}

	gapi.client.load('bmca', 'v1', callback, apiRoot);
};
```

Código 28. Carga de los Endpoints y posteriormente recuperación de todas las actividades

Así las funciones creadas para consumir los servicios disponibles son las siguiente.

- Listar las actividades existentes.

```
google.endpoints.bmcaApi.list = function() {
	gapi.client.bmca.item.list().execute(function(resp) {
		if (resp.items) {
			for (var i = 0; i < resp.items.length; i++) {
				[...]
			}
		}
	});
};
```

Código 29. Código para consumir el Endpoint para listar todas las actividades

- Añadir una nueva actividad.

```
google.endpoints.bmcaApi.add = function() {
	var addCategory = $('[id="addCategory"] :selected').val();
	var addTitle = $('[id="addTitle"]').val();
	var addDescription = $('[id="addDescription"]').val();
	var addAuthor = $('[id="addAuthor"]').val();

	gapi.client.bmca.item.add({'category': addCategory, 'title' : addTitle,
'description': addDescription, 'author' : addAuthor}).execute(function(resp) {
		if (resp) {
			[...]
		}
	});
};
```

Código 30. Servicio para crear una nueva actividad desde el cliente WEB

- Modificar una actividad existente.

```
google.endpoints.bmcaApi.update = function() {
	var updateId = $('[id="updateId"]').val();
	var updateCategory = $('[id="updateCategory"] :selected').val();
	var updateTitle = $('[id="updateTitle"]').val();
	var updateDescription = $('[id="updateDescription"]').val();
	var updateAuthor = $('[id="updateAuthor"]').val();

	gapi.client.bmca.item.update({'id': updateId, 'category': updateCategory,
'title' : updateTitle, 'description': updateDescription, 'author' :
updateAuthor}).execute(function(resp) {
		if (resp) {
			[...]
		}
	});
};
```

Código 31. Modificar una actividad desde el cliente WEB

- Eliminar una actividad.

```
google.endpoints.bmcaApi.remove = function() {
	var id = $('[id="updateId"]').val();

	gapi.client.bmca.item.delete({'id': id}).execute(function(resp) {
		[...]
	});
};
```

Código 32. Consumo del Endpoint para eliminar una actividad del panel

Así finalmente la aplicación está compuesta por las siguientes pantallas que consumen los servicios creados en el servidor.

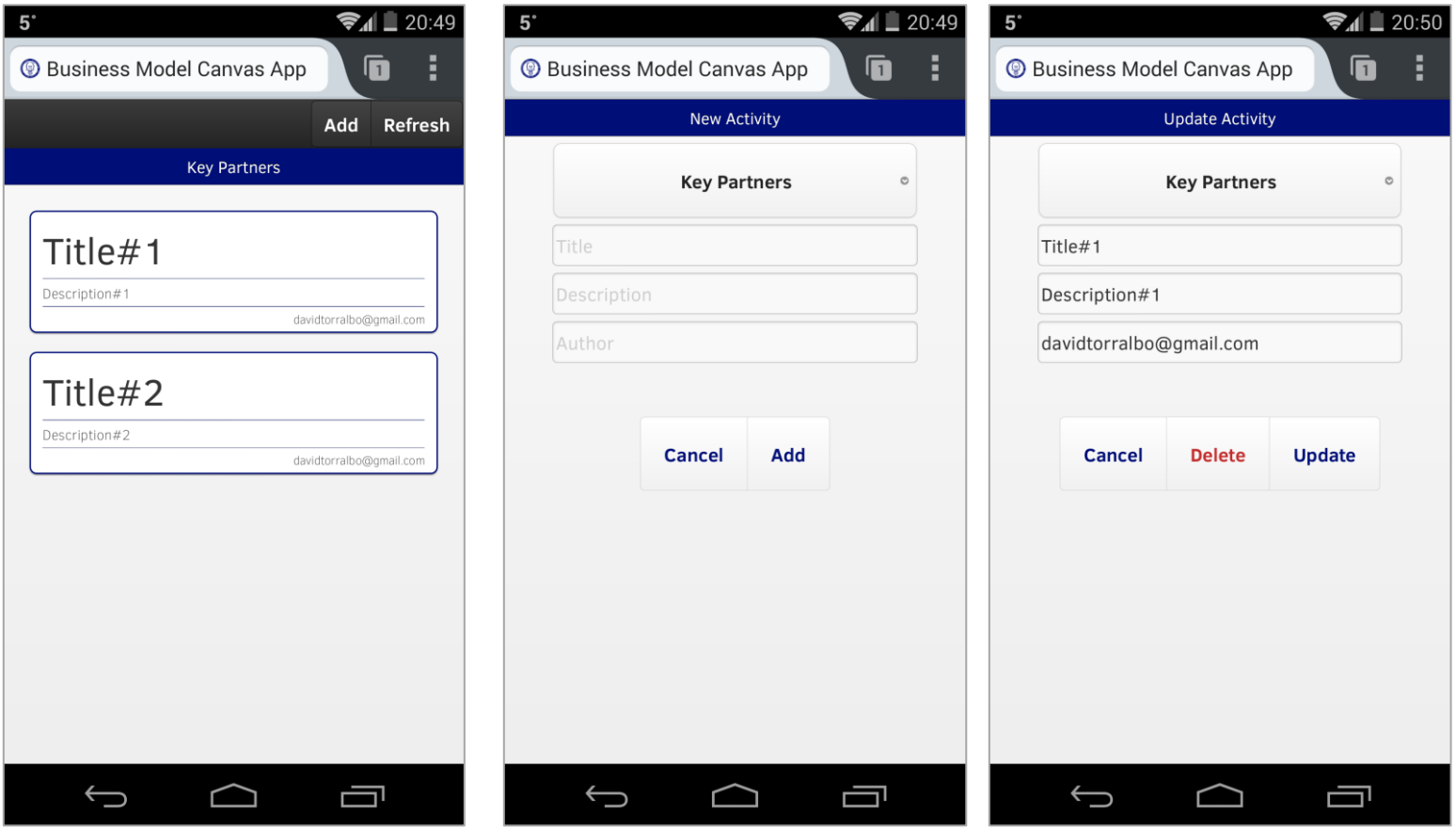


Figura 29. Listar, crear y modificar actividades

3.2.8 Desarrollo del cliente iOS

Google AppEngine y la tecnología *Google Cloud Endpoints* proporcionan las herramientas necesarias para la generación del cliente necesario desde el desarrollo de la aplicación *iOS*.

La generación automática de los *endpoints* durante el desarrollo del *back-end* también proporciona el descriptor necesario para la compilación de la librería. Este descriptor es desplegado en la ruta '*/war/WEB-INF*' del proyecto, en este caso con el nombre '*bmca-v1-rpc.discovery*'. Este descriptor será el necesario para generar la librería siguiendo estos pasos.

- Descargar el proyecto *Google APIs Client Library* proporcionado por *Google*.

```
svn checkout http://google-api-objectivec-client.googlecode.com/svn/trunk/ google-api-objectivec-client-read-only
```

Código 33. Descarga del proyecto Google API Client Library

- Dentro del proyecto descargado, es necesario construir el elemento '*ServiceGenerator.xcodeproj*' (*Project -> Build*).

- Localizar la ruta en la que se encuentra el binario del proyecto *ServiceGenerator*, que debe ser ejecutado sobre el descriptor anteriormente generado.

```
~/Xcode/DerivedData/ServiceGenerator-abc/Build/Products/Debug/ServiceGenerator
~/BusinessModelCanvasApp/server-side/war/WEB-INF/bmca-v1-rpc.discovery --outputDir
~/API
```

Código 34. Ejecución del script ServiceGenerator

Una vez terminado este proceso, se tendrán disponibles todos los ficheros necesarios para poder consumir los servicios creados en *back-end* desde una aplicación *iOS*.

- Generar el servicio con el que se consumirá los *endpoints*.

```
static GTLServiceBmca *service = nil;
if (!service) {
  service = [[GTLServiceBmca alloc] init];
  service.retryEnabled = YES;
  [GTMHTTPFetcher setLoggingEnabled:YES];
}
```

Código 35. Creación del servicio que ofrece los endpoints

- Consumir *endpoints* desde el cliente *iOS*.

```
GTLQueryBmca *query = [GTLQueryBmca queryForItemList];
[service executeQuery:query completionHandler:^(GTLServiceTicket *ticket,
GTLBmcaItem *object, NSError *error) {
  NSArray *items = [object items];
  // Do something with items.}];
```

Código 36. Consumir los endpoint desde el cliente iOS

3.2.9 Desarrollo de una herramienta de logs en el servidor

Con el objetivo de mostrar de una forma más visual la comunicación centralizada entre cada uno de los clientes que colaborativamente consumen los servicios disponibles en el servidor, se decide crear un nuevo servicio que muestre las llamadas realizadas sobre los *endPoints* generados. Este servicio se puede encontrar en la siguiente dirección.

- https://businessmodelcanvasapp.appspot.com/log

Se registran tanto las llamadas para crear una nueva actividad, como las generadas para modificar y para eliminar actividades ya existentes. Así mismo, cada entrada es enriquecida con información no mostrada entre los clientes, como son un identificador único que representa unívocamente a cada actividad y el *user agent* del cliente que ejecuta la petición.

Para que este servicio se mantenga completamente actualizado con cada llamada es necesario un servicio de notificaciones entre el servidor y la herramienta. Se ha decidido utilizar para ello la tecnología *Channel API* que *Google* ofrece. Con esta tecnología se consigue crear y mantener una comunicación en tiempo real entre un cliente *javascript* y un servidor, permitiendo así el envío de mensajes entre ambos actores sin necesidad de hacer *polling*, comunicación no recomendada para la conexión entre distintos componentes situados en internet.

Se aprovecha el mismo servidor *Google App Engine* que se ha utilizado para ser el servidor central de la lógica de negocio de la aplicación como el servidor que corra con este nuevo servicio de logs. Esta tecnología crea una comunicación entre el cliente y el servidor identificándose mediante un *token* que una vez generado en la parte servidora es usado por el cliente para identificar el canal por el que recibir y enviar los mensajes.

Para esto, se crea un nuevo servicio usando la tecnología *Servlet* que se ejecutará cuando el recurso "*/log*" es requerido. Su configuración en el *web.xml* de la aplicación es:

```
<servlet>
	<servlet-name>InitLogServlet</servlet-name>
	<servlet-class>com.dtorralbo.bmca.channel.LogServlet</servlet-class>
</servlet>
<servlet-mapping>
	<servlet-name>InitLogServlet</servlet-name>
	<url-pattern>/log</url-pattern>
</servlet-mapping>
```

Código 37. Configuración del servidor para ejecutar el servlet de log

Esta configuración es adicional a la ya generada automáticamente por el *Plugin de Google para Eclipse* en el momento de generar los *Cloud EndPoints*, por lo que el servidor podrá dar servicio a ambas aplicaciones.

Con este *Servlet* se ejecutará el siguiente código encargado de generar dicho *token* y de transmitirlo al cliente que lo usará en el envío y recepción de mensajes.

```
public class LogServlet extends HttpServlet {

@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
	ChannelService channelService = ChannelServiceFactory.getChannelService();
	String token = channelService.createChannel("bmca_log");

	// Send the token to the client
	FileReader reader = new FileReader("log_template");
	CharBuffer buffer = CharBuffer.allocate(16384);
	reader.read(buffer);
	String board = new String(buffer.array());
	board = board.replaceAll("\\{\\{ token \\}\\}", token);

	reader.close();

	resp.setContentType("text/html");
	resp.getWriter().write(board);
}
}
```

Código 38. Generar un token con el que abrir un canal entre el servidor y la herramienta

Una vez enviado el *token* al cliente, éste debe abrir el canal que enlace a ambos componentes. Para ello se usa la siguiente función *javascript*.

```
openChannel = function() {
	var token = '{{ token }}';
	var channel = new goog.appengine.Channel(token);
	var handler = {
		'onopen': onOpened,
		'onmessage': onMessage,
		'onerror': onError,
		'onclose': onClose
	};
	var socket = channel.open(handler);
	socket.onopen = onOpened;
	socket.onmessage = onMessage;
	socket.onerror = onError;
	socket.onclose = onClose;
}
```

Código 39. Abrir un canal mediante Channel API con el servidor a partir del token recibido

Con esta función se especifican las funciones que a su vez se ejecutarán cuando la conexión ha sido finalmente establecida (*onOpened*), si se produce un error en la conexión (*onError*), al cerrar la comunicación (*onClose*) y cada vez que un mensaje es enviado desde el servidor al cliente (*onMessage*).

La tecnología *Channel API* permite igualmente que el cliente envíe mensajes al servidor, el cual puede generar una notificación *broadcast* al resto de clientes conectados al mismo canal. En este caso no se va a hacer uso de este potencial debido a que este cliente *log* actuará de forma pasiva a los mensajes enviados desde el cliente, como respuesta a los servicios consumidos desde *Android o WebApp.*

Estos mensajes enviados desde el servidor a este cliente son generados gracias a un servicio adicional creado dentro de la lógica principal de negocio, ejecutado cada vez que una actividad es añadida, modificada o eliminada. El mensaje es creado en formato *JSON* y enviado a través del canal generado a partir del *client_id* que identifica al canal.

```
private static final String CLIENT_ID = "bmca_log";

public static void addCanvasItemNotification(Long id, String userAgent) {
	try {
		JSONObject message = new JSONObject();
		message.append("action", "Added");
		message.append("id", id);
		message.append("user_agent", userAgent);

		ChannelService channelService =
ChannelServiceFactory.getChannelService();
		ChannelMessage channelMessage = new ChannelMessage(CLIENT_ID,
message.toString());

		channelService.sendMessage(channelMessage);

	} catch (JSONException e) {
		e.printStackTrace();
	}
}
```

Código 40. Enviar mensaje del servidor a la herramienta de log

Este servicio pues es ejecutado desde de los *EndPoints* cuando se añade una nueva actividad, simplemente es necesario añadir esta lógica en el *endPoint* correspondiente.

```
@ApiMethod(name = "Item.add")
public CanvasItem insertCanvasItem(CanvasItem canvasitem, HttpServletRequest
request) {
	// Send message to the log service
	LogService.addCanvasItemNotification(canvasitem.getId(),
request.getHeader("User-Agent"));

	return canvasitem;
}
```

Código 41. Servicio ejecutado desde los endpoints para mostrar las actividades en la herramienta de log

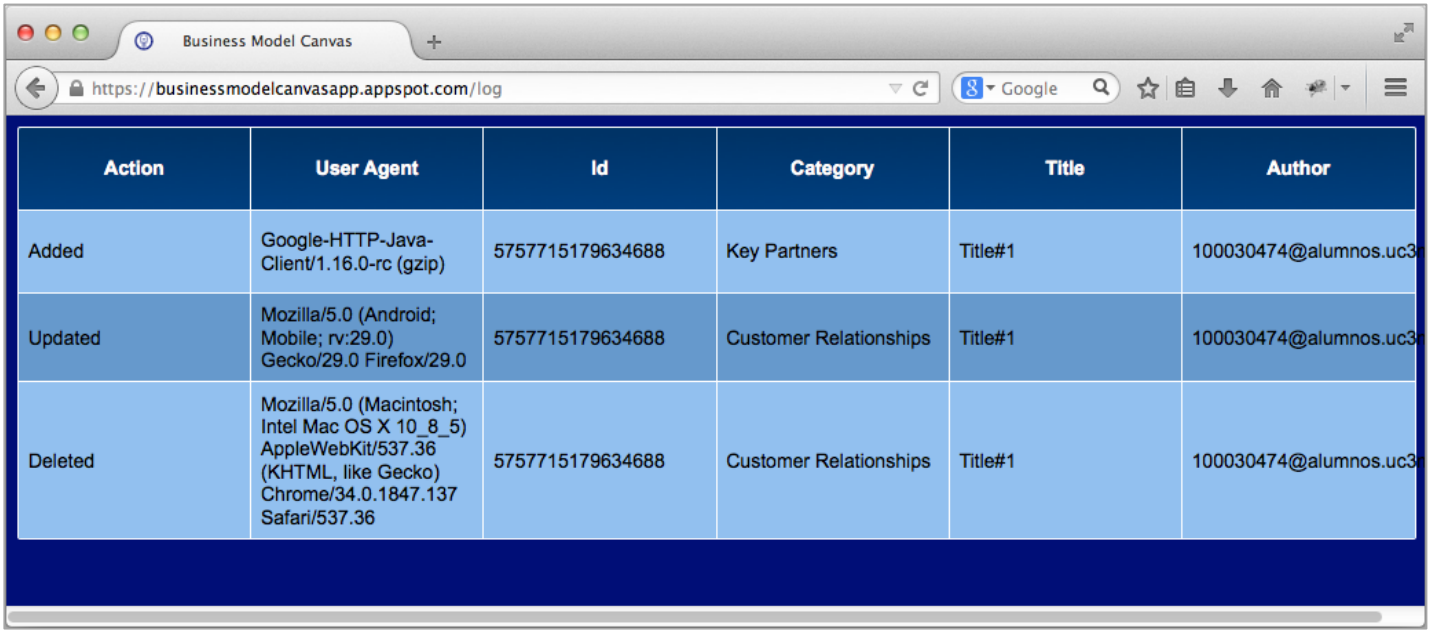
Es importante destacar que este servicio generado automáticamente por la tecnología *Goolge Cloud EndPoints* no proporciona por defecto el objeto *HttpServletRequest* utilizado para obtener el *user agent* del cliente, pero debido a la naturaleza de estos *EndPoints* (*Servlets*) solo es necesario declarar el objeto como un parámetro más del servicio y automáticamente será añadido en la llamada a dicho método.

Finalmente una vez el mensaje se ha enviado desde el servidor a la herramienta *log* se ha decidido que haga uso igualmente de los *endPoints* disponibles para obtener la información relevante a la actividad que, en este caso, ha sido añadida. Para lo cual se utiliza el servicio *bmca.item.get(id)*.

```
onMessage = function(m) {
	message = JSON.parse(m.data);
	action = message.action;
	if (action == "Added") {
		google.endpoints.bmcaApi.added(message);
	}
}
google.endpoints.bmcaApi.added = function(message) {
	var id = message.id;
	var action = message.action;
	gapi.client.bmca.item.get({'id' : id[0]}).execute(function(resp) {
		if (resp) {
			google.endpoints.bmcaApi.log(action, resp, message.user_agent);
		}
	});}
google.endpoints.bmcaApi.log = function(action, resp, user_agent) {
	var canvasItem = document.createElement('tr');
	canvasItem.innerHTML = "<td>" + action + "</td><td>" + user_agent+
"</td><td>" + resp.id + "</td><td>" + resp.category + "</td><td>"+ resp.title +
"</td><td>" + resp.author + "</td>";
	document.getElementById("log_table").appendChild(canvasItem);
}
```

Código 42. Lógica para mostrar una nueva entrada en la herramienta

Así la herramienta es capaz de mostrar información sobre las acciones principales que los clientes *Android y Web* han ejecutadp. Se registran incluso acciones ejecutadas desde la herramienta *API Explorer*.



Action	User Agent	Id	Category	Title	Author
Added	Google-HTTP-Java-Client/1.16.0-rc (gzip)	5757715179634688	Key Partners	Title#1	100030474@alumnos.uc3
Updated	Mozilla/5.0 (Android; Mobile; rv:29.0) Gecko/29.0 Firefox/29.0	5757715179634688	Customer Relationships	Title#1	100030474@alumnos.uc3
Deleted	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.137 Safari/537.36	5757715179634688	Customer Relationships	Title#1	100030474@alumnos.uc3

Figura 30. Visualización de los servicios consumidos desde los clientes

3.2.10 Desarrollo de un panel principal en el servidor

Aunque el desarrollo original de este proyecto es el de una aplicación móvil multiplataforma, también es posible hacer uso del potencial de los *endPoints* desde una aplicación *desktop* usando el cliente *javascript* generado.

Para mostrar así la interacción entre los distintos clientes desarrollados se decide crear un panel central que muestre las actividades existentes y el resultado de las distintas acciones desde los clientes *Android y Web.* Es posible acceder a este panel a través de la siguiente dirección.

- https://businessmodelcanvasapp.appspot.com

Se hace uso de la tecnología *Channel API de Google* para notificar al panel en tiempo real de los cambios realizados en las actividades, tanto al añadir una nueva actividad como al modificar o eliminar una ya existente. Para ello debe crearse un canal mediante el cual el servidor enviará los mensajes al panel, tal y como se ha hecho en la herramienta de logs.

Se configura un *Servlet* para que se ejecute cuando el recurso del panel es solicitado, dentro del cual se crea el *token* que es enviado al cliente y que representará al canal de comunicación.

```
<welcome-file-list>
	<welcome-file>board</welcome-file>
</welcome-file-list>

<servlet>
	<servlet-name>InitBoardServlet</servlet-name>
	<servlet-class>com.dtorralbo.bmca.channel.BoardServlet</servlet-class>
</servlet>
<servlet-mapping>
	<servlet-name>InitBoardServlet</servlet-name>
	<url-pattern>/board</url-pattern>
</servlet-mapping>
```

Código 43. Configuración de un nuevo servlet para el panel central en el servidor

Con la configuración del *welcome-file list* se consigue además que el recurso por defecto de este site sea el de este panel, por lo cual solo es necesario acceder al dominio principal para tener acceso a él:

- *https://businessmodelcanvasapp.appspot.com*

El *servlet* crea el canal y genera el token de la misma forma que se ha hecho en la herramienta de logs.

```
public class BoardServlet extends HttpServlet {

@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {

	ChannelService channelService = ChannelServiceFactory.getChannelService();
	String token = channelService.createChannel("bmca_board");

	FileReader reader = new FileReader("board_template");
	CharBuffer buffer = CharBuffer.allocate(16384);
	reader.read(buffer);
	String board = new String(buffer.array());
	board = board.replaceAll("\\{\\{ token \\}\\}", token);

	reader.close();

	resp.setContentType("text/html");
	resp.getWriter().write(board);
}
}
```

Código 44. Creación del token con el que crear el canal entre el servidor y el panel

Este *token* es usado en el cliente para abrir la comunicación con el servidor a través del canal que previamente se ha creado, así una vez renderizado el panel se ejecuta el siguiente código *javascript*.

```
<script type="text/javascript">
	openChannel = function() {
		var token = '{{ token }}';
		var channel = new goog.appengine.Channel(token);
		var handler = {
			'onopen': onOpened,
			'onmessage': onMessage,
			'onerror': onError,
			'onclose': onClose
		};
		var socket = channel.open(handler);
		socket.onopen = onOpened;
		socket.onmessage = onMessage;
		socket.onerror = onError;
		socket.onclose = onClose;
	}
	[...]
</script>
```

Código 45. Apertura del canal desde el panel a partir del token recibido

Una vez el panel ha abierto la conexión con el servidor ya está disponible para recibir los mensajes del mismo. En este caso además el panel debe mostrar las actividades existentes en la aplicación y calificarlas a partir de su categoría, esta acción se realizará accediendo a los servicios que los *endPoints* tienen disponibles.

Para ello, una vez se ha terminado de realizar la conexión mediante *Channel API*, se cargan los *endPoints* del servidor y se accede al servicio *gapi.client.bmca.item.list()* con el que se obtendrán todas las actividades existentes.

```
<script>
	[...]

	initialize = function() {
		openChannel();
		google.endpoints.bmcaApi.init('/_ah/api');
	}

</script>
<script src="https://apis.google.com/js/client.js?onload=initialize"></script>
```

Código 46. Apertura del canal e inicialización de los endpoints para poder listas las actividades existentes

```
google.endpoints.bmcaApi.init = function(apiRoot) {
	var callback = function() {
		google.endpoints.bmcaApi.list();
	}
	gapi.client.load('bmca', 'v1', callback, apiRoot);
};

google.endpoints.bmcaApi.list = function() {
	gapi.client.bmca.item.list().execute(function(resp) {
		if (resp.items) {
			for (var i = 0; i < resp.items.length; i++) {
				var canvasItem = document.createElement('div');
				canvasItem.id = resp.items[i].id;
				canvasItem.className = "canvasItem";
				canvasItem.innerHTML = "<div class=\"title\">"+
resp.items[i].title +"</div><hr><div class=\"description\">"+
resp.items[i].description +"</div><hr><div class=\"author\">"+ resp.items[i].author
+"</div>";
	document.getElementById(resp.items[i].category).appendChild(canvasItem);
			}
		}
	});
};
```

Código 47. Apertura del canal e inicialización de los endpoints para poder listas las actividades existentes

En este punto el panel ya ha mostrado todas las actividades y está listo para recibir los mensajes del servidor sobre la creación, modificación o eliminación de las actividades. Estas notificaciones son realizadas gracias a un servicio adicional que es ejecutado cuando una de estas acciones es realizada. Este servicio *BoardUpdateService* será el encargado de mandar los mensajes correspondientes a la acción ejecutada.

```
public class BoardUpdateService {
	private static final String CLIENT_ID = "bmca_board";

	public static void addCanvasItemNotification(Long id) {
		try {
			JSONObject message = new JSONObject();
			message.append("action", "Added");
			message.append("id", id);

			ChannelService channelService =
ChannelServiceFactory.getChannelService();
			ChannelMessage channelMessage = new ChannelMessage(CLIENT_ID,
message.toString());
			channelService.sendMessage(channelMessage);
		} catch (JSONException e) {e.printStackTrace();}
}}
```

Código 48. Servicio en el servidor para enviar mensajes al panel

Y será consumido desde los *endPoints* cuando los clientes realicen alguna acción sobre las actividades.

```
@ApiMethod(name = "Item.add")
public CanvasItem insertCanvasItem(CanvasItem canvasitem, HttpServletRequest
request) {
	[...]

	BoardUpdateService.addCanvasItemNotification(canvasitem.getId());

	[...]
}
```

Código 49. Llamada desde los endpoints a los nuevos servicios

Así cuando una nueva actividad es creada por un cliente, se ejecuta este servicio *BoardUpdateService.addCanvasItemNotification(Long id);*. Este servicio se encargará de enviar un mensaje al panel indicando el id único de la nueva actividad, gracias al cual y a los *endPoints* que se han generado en la lógica de negocio se accede al resto de información que será mostrada en el panel.

```
onMessage = function(m) {
	message = JSON.parse(m.data);
	action = message.action;
	if (action == "Added") {
		google.endpoints.bmcaApi.added(message);
	} else if (action == "Updated") {
		google.endpoints.bmcaApi.updated(message);
	} else if (action == "Deleted") {
		google.endpoints.bmcaApi.deleted(message);
	}
};

google.endpoints.bmcaApi.added = function(message) {
	var id = message.id;
	gapi.client.bmca.item.get({'id' : id[0]}).execute(function(resp) {
		if (resp) {
			var canvasItem = document.createElement('div');
			canvasItem.id = resp.id;
			canvasItem.className = "canvasItem";

			canvasItem.innerHTML = "<div class=\"title\">"+ resp.title
+"</div><hr><div class=\"description\">"+ resp.description +"</div><hr><div
class=\"author\">"+ resp.author +"</div>";
			document.getElementById(resp.category).appendChild(canvasItem);

		}
	});
};
```

Código 50. Lógica en el panel para añadir una nueva actividad al panel

De la misma forma se tratan las acciones de modificar y eliminar actividades. Cuando el cliente ejecuta alguna de los *endPoints* el servicio de notificación del panel envía un mensaje con el id de la actividad con el cual se consume en servicio *bmca.item.get(id)* de los *endPoints*.

El panel finalmente queda de la siguiente forma.

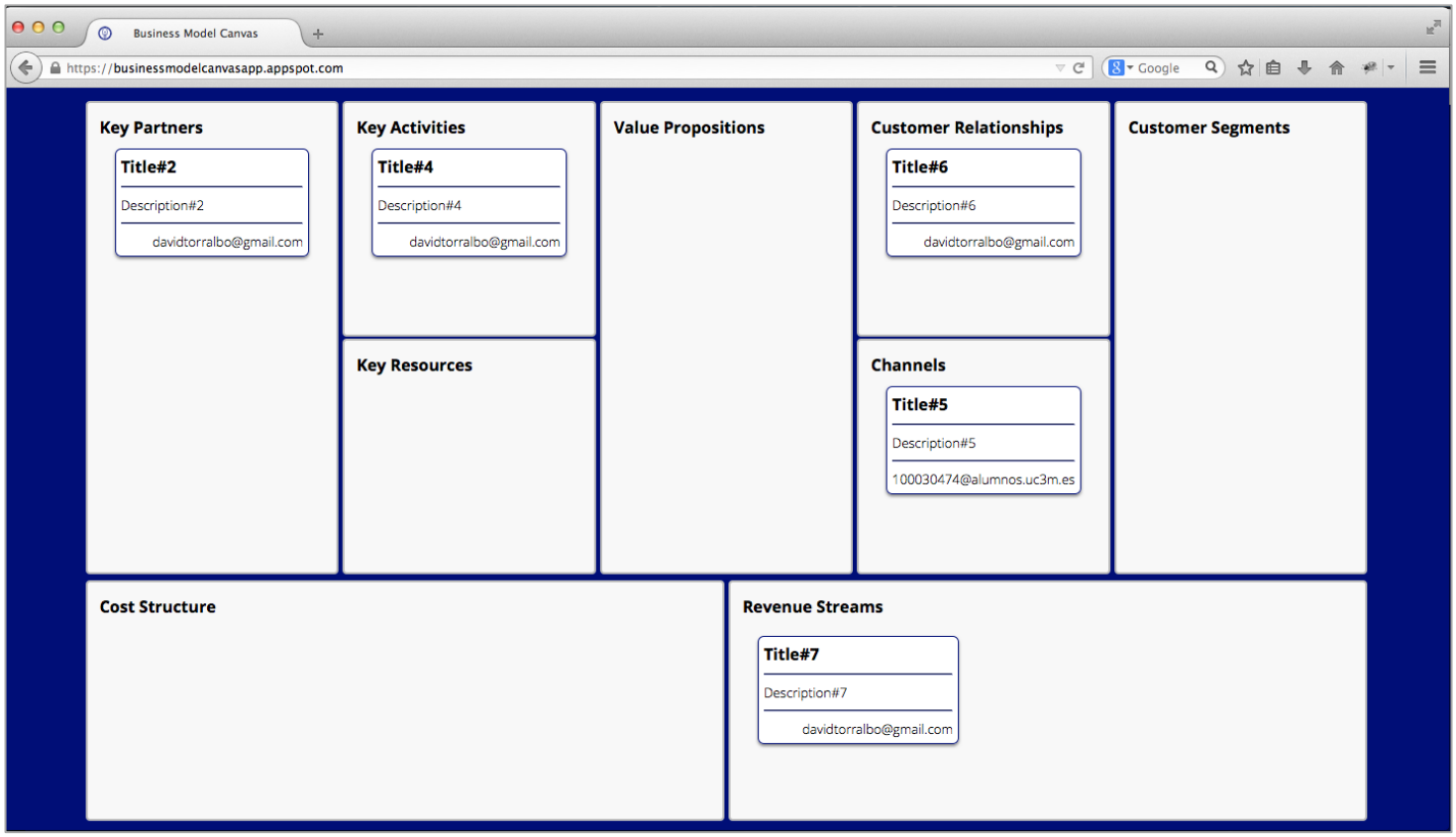


Figura 31. Panel Principal en el servidor

3.3 Complementos y mejoras a este desarrollo

El desarrollo de esta infraestructura da como resultado un sistema sencillo mediante el cual clientes externos pueden consumir una lógica alocada en un servidor central. Este sistema a priori es funcional y puede ser consumido tal y como se ha expuesto. Pero existen algunas líneas de mejora algunos complementos al desarrollo que harían de la aplicación un producto más completo y con más valor añadido.

3.3.1 Configuración de un sistema de seguridad adicional. OAuth 2.0

Google Cloud Endpoints soporta el autenticado basado en *OAuth 2.0* mediante el cual es posible delegar el autenticado de usuarios a sistemas externos. Con ello es posible hacer uso de terceros clientes para la gestión de los usuarios. Con ello no solo se puede simplificar en gran medida la gestión de las entidades de los usuarios sino que a su vez es posible aprovecharse de sistemas de autenticación más comunes y conocidos entre los usuarios, como pueden ser *Google+, Facebook, Twitter,* etc.

Específicamente, *Google Cloud Endpoints* proporciona una integración inmediata con el sistema de autenticación de *Google+*, haciendo uso de su plataforma y sus credenciales para identificar y logar a un usuario en el sistema que se está desarrollando.

Gracias a la herramienta *Google Developer Console* se pueden generar los *clientId* necesarios para permitir el acceso a los servicios disponibles desde los clientes *Android, WEB y iOS*. Igualmente, estos *clientId* deben ser configurados en el servidor de *back-end* para abrir el acceso a aquellos clientes que usen estas credenciales. Para ello simplemente hay que añadir a la configuración inicial de los *endpoints* la lista de clientes y sus audiencias.

```
@Api(
    name = "bmca",
    version = "v1",
    scopes = {Constants.EMAIL_SCOPE},
    clientIds = {Constants.WEB_CLIENT_ID, Constants.ANDROID_CLIENT_ID, Constants.IOS_CLIENT_ID},
    audiences = {Constants.ANDROID_AUDIENCE}
)

public class Constants {
  public static final String WEB_CLIENT_ID = "1-web-apps.apps.googleusercontent.com";
  public static final String ANDROID_CLIENT_ID = "2-android-apps.googleusercontent.com";
  public static final String IOS_CLIENT_ID = "3-ios-apps.googleusercontent.com";
  public static final String ANDROID_AUDIENCE = WEB_CLIENT_ID;
  public static final String EMAIL_SCOPE = "https://www.googleapis.com/auth/userinfo.email";
}
```

Código 51. Configuración para el uso de OAuth 2.0 en los endpoints

Una vez los *endpoints* han sido configurados para aceptar peticiones de clientes configurados con estas credenciales, es posible acceder a los usuarios que acceden a los servicios una vez se han logado mediante *OAuth 2.0 de Google+*. Para ello simplemente hay que añadir el objeto *User* como nuevo parámetro de aquellos servicios que se definan deben ser ejecutados por usuarios autenticados.

```
import com.google.appengine.api.users.User;

@ApiMethod(name = "Item.add")
public CanvasItem insertCanvasItem(CanvasItemcanvas item,
                                   HttpServletRequest request,
                                   User user) {

[...]

}
```

Código 52. Objeto User necesario en OAuth 2.0

No es necesario que todos los servicios sean consumidos por clientes que han realizado el proceso de autenticación, por ejemplo el servicio para listar todas las actividades existentes puede ser ejecutado sin necesidad de estar logado en el sistema.

Una vez añadido el objeto *User* como parámetro del servicio, es necesario realizar algunas comprobaciones de seguridad, como por ejemplo que dicho objeto no está vacío, lo que representaría que se está accediendo a dicho servicio con un usuario no autenticado.

Alguna de la información más relevante que este objeto *User* proporciona puede ser:

```
// Email of the user
user.getEmail();

// Name of the user
user.getNickname();

// Domain of the authentication of the user
user.getAuthDomain();
```

Código 53. Funciones más destacables disponibles sobre el objeto User

A partir de estos datos no solo es posible acotar el acceso a los servicios de usuarios autenticados sino que además se puede tanto añadir lógica particular a los usuarios no logados como otorgar funcionalidades diferentes dependiendo de los roles de los usuarios.

3.3.2 Comunicación Servidor - Cliente

En este proyecto se han estudiado las diferentes tecnologías para el desarrollo de una aplicación cliente que pueda consumir servicios alojados en un servidor central, y principalmente la tecnología *Google Cloud Endpoints* como solución recomendada para ello. Pero la comunicación en este desarrollo se realiza desde la aplicación cliente hacia el servidor compartido.

Una de las mejoras propuestas para este desarrollo es la implementación de una comunicación Cliente - Servidor, mediante la cual las diferentes interacciones de los usuarios y las modificaciones realizadas puedan ser notificadas a todos los clientes conectados al sistema en cada momento. Básicamente se debería realizar una notificación *broadcast* a los clientes que estén ejecutando la aplicación.

Se han analizado los clientes *Android, WEB e iOS* pormenorizado y en este caso la solución para esta comunicación también puede analizarse por separado.

- **Android**: Google Cloud Message

Este servicio ofrecido por *Google* permite enviar mensajes desde un servidor a aplicaciones *android* conectadas al mismo [14]. Estos mensajes pueden ser desde simples mensajes de notificación sobre cambios en el servicio hasta mensajes más complejos de hasta 4kb de datos.

El potencial principal de este servicio son sus capacidades de encolamiento y entrega de mensajes, con los cuales es posible abstraerse del desarrollo de los mismos haciendo la comunicación más sencilla para los desarrolladores.

Además, debido al acomplamiento de las aplicaciones *Android* con los sistemas de *Google*, la aplicación no tiene que estar en ejecución en el momento de la recepción del mensaje. El propio mensaje generará una notificación *broadcast* que inicializará la aplicación que deba consumir cada mensaje.

- **WEB**: Channel API

Esta tecnología, también proporcionada por *Google*, está basada en una comunicación estándar mediante *Sockets* [15]. Con ella se crea una comunicación permanente entre cl cliente *WEB* y el servidor central el cual podrá enviar mensajes a todos aquellos clientes que se encuentren enlazados con el canal creado.

El servidor podrá enviar información a clientes *javascript* en tiempo real sin la necesidad de hacer llamadas continuas al servidor (o *polling*). Esto proporciona una nueva vía de comunicación entre los componentes mejorando el rendimiento y los recursos del sistema.

En esta comunicación, el cliente crea un identificador único con el cual se crea un canal o *channel* que representa la conexión entre el cliente y el servidor. De esta forma es posible enviar mensaje de forma personalizada a cada cliente.

- **iOS**: Apple Push Notification

Servicio proporcionado por *Apple* con el que propagar información a terminales *iOS y OS X* [16]. En este caso, los clientes *iOS* crean una conexión acreditada y encriptada basada en la IP del terminal, mediante la cual enviar información sobre la conexión establecida.

Al igual que ocurre para los clientes *Android*, si una notificación llega al terminal cuando la aplicación que debe consumirla no está en ejecución, el terminal alerta al usuario de que existe información disponible para ser consumida por dicha aplicación.

Este servicio proporciona unos requisitos de calidad mínimos que se caracterizan principalmente por su capacidad de *store-and-forward* de mensajes, lo que permite almacenar un mensaje que no ha sido posible enviarse para su reintento posterior.

Dado que el sistema operativo *iOS* no es un sistema por defecto integrado en las diferentes tecnologías *Google* que en este proyecto se están estudiando, *Google App Engine* proporciona la capacidad de usar este servicio *Apple Push Notification* desde sus servidores [17], con lo cual el uso de *App Engine* para esta solución es igualmente aceptable.

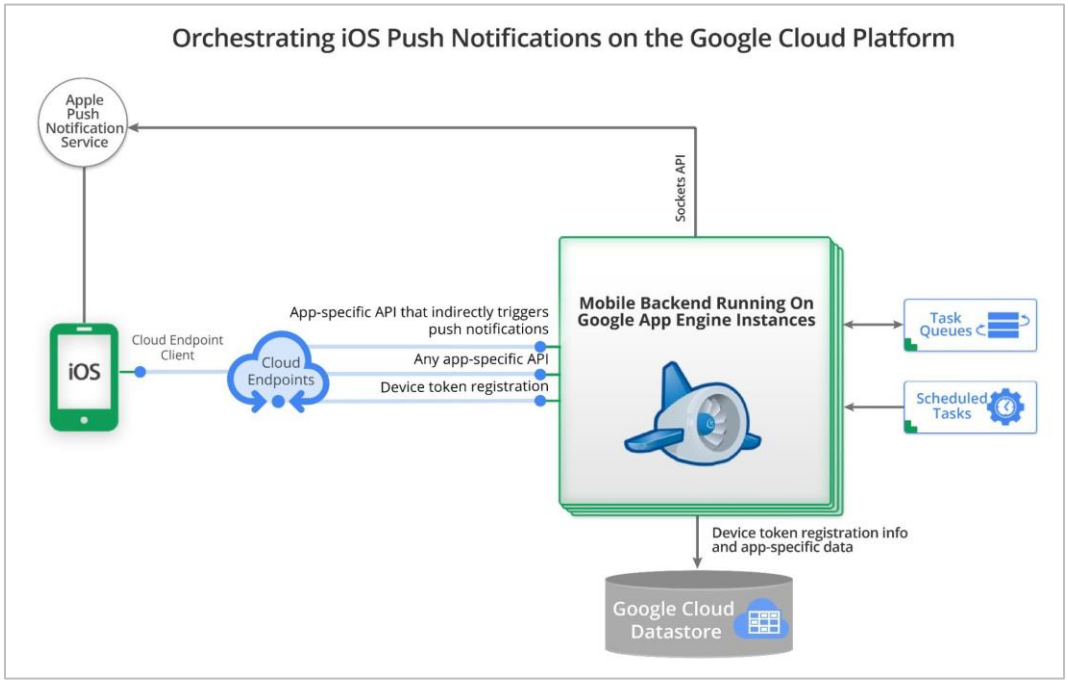


Figura 32. Esquema de comunicación entre App Engine y Apple Push Notification

3.3.3 Consumo de los servicios desde otras plataformas móviles

Uno de los grandes potenciales de este desarrollo es que se trata de un sistema multiplataforma, y aunque la tecnología *Google Cloud Endpoints* dé soporte dedicado a clientes desarrollados bajo *Android, iOS y WEB (Javascript)*, es posible acceder a los mismo servicios desde otras plataformas de desarrollo de aplicaciones móviles.

- Firefox OS

Este cliente en sí se basa en la tecnología *javascript* y es posible hacer uso del cliente *WEB* ya desarrollado anteriormente para crear una aplicación completamente funcional dentro del desarrollo de *Firefox OS*. Para ello solo es necesario añadir un descriptor propio para que el sistema operativo *Firefox OS* pueda ejecutar dicho cliente.

Este descriptor debe ser emplazado dentro de la aplicación *WEB* cliente (*/BusinessModelCanvasApp/web-client/war*) con nombre '*manifest.webapp*'.

```
{
  "name": "BMCA",
  "description": "Business Model Canvas App",
  "launch_path": "/index.html",
  "icons": {
    "60": "img/ic_launcher_60.png",
    "128": "img/ic_launcher_128.png"
  },
  "developer": {
    "name": "David Torralbo",
    "url": "https://github.com/Sca09/BusinessModelCanvasApp"
  },
  "default_locale": "en",
  "orientation": "portrait-primary",
  "type": "web",
  "permissions": {
    "systemXHR": {
      "description": "Connection to BMCA server through Google Cloud EndPoints"
    }
  }
}
```

Código 54. Descriptor para definir una aplicación en Firefox OS

Con este descriptor es posible ejecutar este cliente desde *Firefox OS* como aplicación del sistema operativo.

- Blackberry

Al igual que en el caso anterior, es posible utilizar la compatibilidad de *Blackberry* con aplicaciones *Android* para igualmente poder consumir los servicios generados desde otras plataformas móviles. Esta compatibilidad hace que sea posible instalar aplicaciones *Android* en el sistema operativo *Blackberry* y ejecutarlas como propias.

Además, el lenguaje recomendado para el desarrollo de aplicaciones *Blackberry* es HTML5 por lo que el acceso a los servicios también podría hacerse mediante los clientes *javascript* que igualmente se proporcionan.

- Otras plataformas

Como se ha visto a lo largo de este proyecto, la tecnología *Google Cloud Endpoint* principalmente se encarga de facilitar el desarrollo de los mecanismos de comunicación entre cliente y servidor a través de internet. Para ellos encapsula la lógica necesaria para realizar peticiones a través de la red.

Pero como se ha visto igualmente, estos servicios tienen naturaleza *RESTFul*, lo que proporciona el potencial de consumir estos servicios accediendo de forma tradicional a esos *endpoint RESTFul*.

3.3.4 Google Cloud Endpoint y los Wearables

La tecnología *Google Cloud Endpoints* está concebida para facilitar la comunicación entre el servidor y sus clientes, potencialmente clientes móviles, entre los cuales ya es posible considerar los nuevos dispositivos *Wearables*. Desde *Google* se ha apostado de forma significativa en los últimos años por estos dispositivos, tanto es así que en el *Google I/O* del año 2012 se lanzó el *Google Glass* y en el *Google I/O* del año 2014 la temática central eran los *Smart Watches*.

Igualmente estos dispositivos pueden hacer uso de los *Google Cloud Endpoints* con el fin de comunicarse con el servidor. Ambos dispositivos, haciendo uso de la plataforma *Android Wear*, usarán el teléfono móvil como punto de comunicación con internet, y es por ello por lo que pueden aprovecharse de lo que los *Google Cloud Endpoints* pueden ofrecer. El dibujo final quedaría de la siguiente forma.

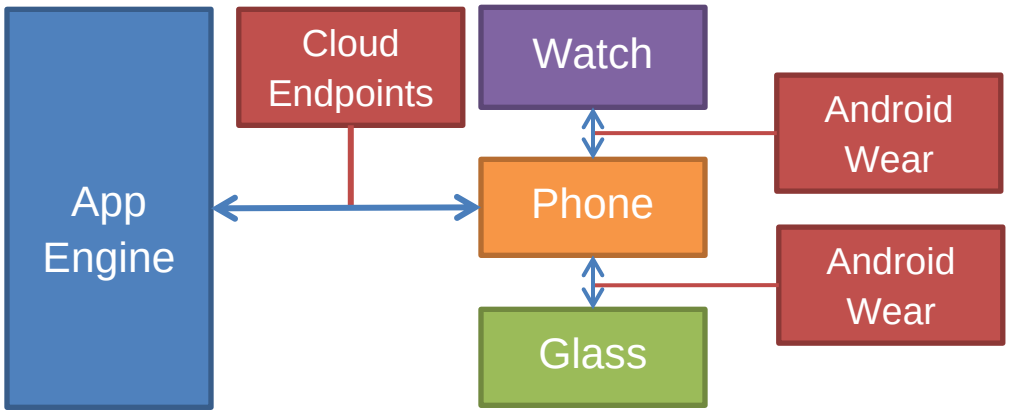


Figura 33. Comunicación entre App Engine y dispositivos Wearables mediante Google Cloud Endpoints

Capítulo 4

Tiempo y Presupuesto

4.1 Metodología y control de tareas

Para el diseño y desarrollo de la aplicación final se decidió hacer uso de la metodología *Agile* y más concretamente el modelo *Scrum* para la definición y control de tareas. Como herramienta de apoyo se ha utilizado *Trello* con el cual definir y manejar tareas dentro de la metodología.

Gracias a esta herramienta, disponible tanto en formato *WEB* como en formato *Android*, se ha podido seguir un control del tiempo y de las tareas a realizar en cada *Sprint* en todo momento, manteniéndose tanto el tutor como el alumno completamente sincronizados.

Esto ha sido de gran ayuda al trabajo en remoto que durante todo el proyecto se ha mantenido entre ambos colaboradores.

.

Figura 34. Panel Trello en formato WEB para el seguimiento de las tareas mediante la metodología Scrum

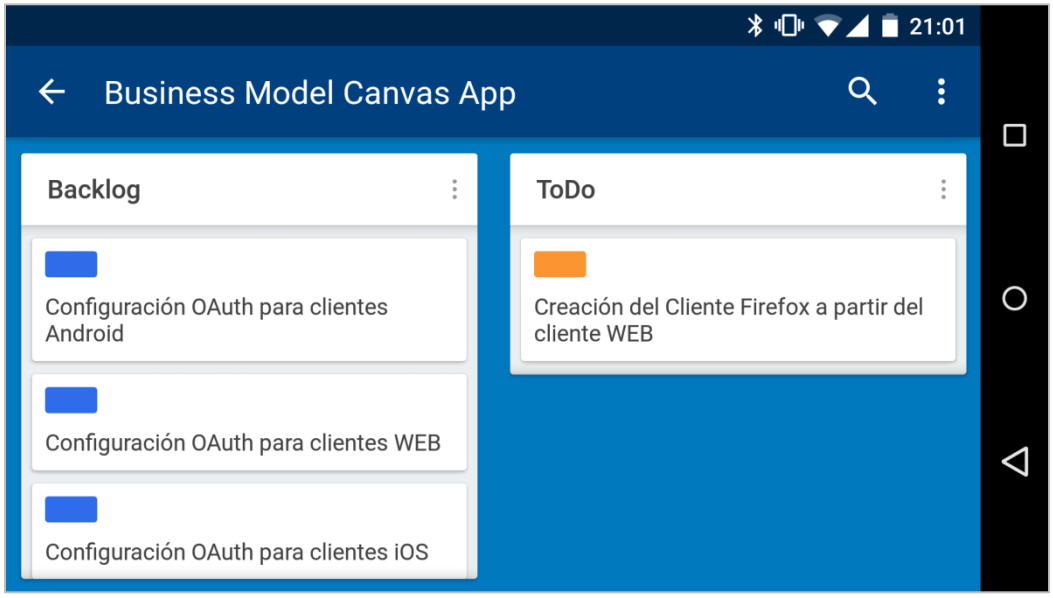


Figura 35. Trello en formato Android

4.2 Desglose del tiempo invertido

La inversión de tiempo en el desarrollo del ejemplo propuesto se puede detallar de la siguiente forma.

Sprint	Tarea	Tiempo (horas)
1	Creación de un nuevo repositorio en GITHUB	0,5
	Creación del servidor Google App Engine en local	1
	Definición del modelo de Datos en formato JDO	1
	Generación de la lógica de negocio base vía Google Cloud EndPoints	0,5
	Implementación de lógica de negocio adicional	1
	Implementación de herramienta de logs en servidor	6
	Despliegue del servidor a Google App Engine	0,5
2	Creación de un panel principal sobre el que añadir actividades desde las aplicaciones	5
	Creación de servidor para Cliente WEB (Google App Engine temporal) en local	0,5
	Implementación de la estructura del Cliente WEB	12
	Consumo de la lógica de negocio mediante librería cliente js de Google Cloud Endpoints	4
	Despliegue del Cliente WEB a Google App Engine (temporal)	0,5
3	Implementación de la estructura del Cliente Android	16
	Consumo de la lógica de negocio mediante librería cliente Android de Google Cloud Endpoints	3
	Total	**51**

Tabla 3. Tiempo invertido en el desarrollo del BMCA

Para el análisis de estos tiempos es importante detallar que el desarrollo no ha sido a tiempo completo, por lo que se ha decidido ponderarlos con un factor 4:1, es decir, se ha invertido diariamente en este proyecto un cuarto de lo que se considera una jornada laboral estándar.

Tarea	Tiempo
Definición del Proyecto	3 semanas
Definición de requisitos funcionales	2 semanas
Estudio del Arte	6 semanas
Sprint #1 - Lógica de negocio	1 semana
Sprint #2 - Cliente WEB	2 semanas
Sprint #3 - Cliente Android	2 semanas
Documentación	4 semanas
Total	**20 semanas**

Tabla 4. Tiempo empleado en cada fase del proyecto

Por lo cual, si se extrapola este tiempo invertido a una jornada laboral estándar, el total del tiempo necesario sería el siguiente.

Tarea	Tiempo
Total	**5 semanas**

Tabla 5. Tiempo real necesario para el desarrollo del proyecto

Esto quiere decir que el tiempo invertido es de 25 días laborables, lo que hace un total de 200 horas de trabajo efectivo.

4.3 Coste del Proyecto

Concepto	Coste
Costes Materiales (ordenador)	1.229,00€
Costes de recursos humanos (50€/h)	10.000,00€
Coste Total	**11.229,00€**

Tabla 6. Coste total del proyecto

Referencias

[1] http://www.interoute.com/cloud-article/what-cloud-computing
[2] http://en.wikipedia.org/wiki/Cloud_computing
[3] http://www.networkworld.com/supp/2012/enterprise2/040912-ecs-iaas-companies-257611.html
[4] http://en.wikipedia.org/wiki/Google_App_Engine
[5] http://en.wikipedia.org/wiki/Amazon_EC2
[6] http://www.w3schools.com/webservices/
[7] http://www.nongnu.org/cvs/
[8] http://subversion.apache.org/
[9] http://git-scm.com/
[10] http://nvie.com/posts/a-successful-git-branching-model/
[11] http://www.businessmodelgeneration.com/canvas/bmc
[12] http://fivewhys.wordpress.com/2012/05/22/business-model-innovation/
[13] http://javiermegias.com/blog/2011/11/herramientas-el-lienzo-de-modelos-de-negocio-business-model-canvas/
[14] https://developer.android.com/google/gcm/gcm.html
[15] https://cloud.google.com/appengine/docs/java/channel/
[16] https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html#//apple_ref/doc/uid/TP40008194-CH100-SW9
[17] https://cloud.google.com/developers/articles/ios-push-notifications/
[18] https://cloud.google.com/appengine/docs/java/search/#Java_Search_API_quotas
[19] https://cloud.google.com/appengine/pricing#Billable_Resource_Unit_Costs
[20] http://aws.amazon.com/ec2/instance-types/
[21] http://aws.amazon.com/ec2/pricing/
[22] http://db.apache.org/jdo/jdo_v_jpa.html
[23] https://support.cdmon.com/entries/24127898-Manual-b%C3%A1sico-de-GIT
[24] http://cloud-computing.softwareinsider.com/saved_compare/Amazon-EC2-vs-Google-App-Engine
[25] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[26] https://www.eclipse.org/downloads/

Anexo I

Estudio del Arte: Google App Engine

Ofrece un completo set de servicios y recursos para el desarrollo completo de aplicaciones web.

- *Memcache* para el manejo de caches.
- *Datastore* o bases de datos.
- *URL Fetch* con el que acceder a otros servicios en otras direcciones.
- *Mail*: para el envío de mensajes.
- *XMPP* para mensajería instantánea.
- *Task Queue* para la programación de tareas.
- Tratamiento de imágenes.
- *Blobstore* para el tratamiento de ficheros de gran tamaño.

Como *Paas* por definición, este servicio enmascara gran parte del trabajo necesario para la configuración y mantenimiento de los servidores, y es de destacar la facilidad con la que se encuentran los desarrolladores para poner en funcionamiento su aplicación. Todo esto gracias al modelo de despliegue que *Google App Engine* ofrece junto a las herramientas que están disponibles para ello, como:

- *Google App Engine Launcher*: con el que fácilmente se puede desplegar un servicio web.
- *SDK Console*: herramienta con la que tendremos un gran amplio control sobre la instalación, tanto en un entorno local como en uno remoto.
- *Google Plugin for Eclipse*: paquete de funcionalidades añadidas con las que complementar las ya presentes en el *IDE Eclipse*, con el que el despliegue a un servidor público de la aplicación web es cuestión de un solo *click*.

GAE igualmente proporciona gran cantidad de herramientas dentro de su *plugin* para *Eclipse* con las cuales poder desarrollar en un ambiente de trabajo local completamente controlable por los desarrolladores. Este entorno de trabajo junto con el control de versiones que *App Engine* provee facilita enormemente el desarrollo y reduce considerablemente los puntos de error a la hora de desplegar los nuevos desarrollos a un servidor público.

En cuanto a los costes del servicio es de destacar que el uso primario de *App Engine* no conlleva coste alguno por parte del usuario. La gran mayoría de los servicios que *Google* pone a disposición de sus clientes desarrolladores son de coste gratuito bajo unas cuotas de uso bastante considerable. Con esto es posible realizar un desarrollo básico y experimental de la aplicación final sin enfrentarse a inversiones económicas iniciales. Además ofrecen gratuitamente el despliegue de hasta 10 aplicaciones simultáneas por usuario.

Las cuotas de algunos de los servicios básicos más destacables son:

- Almacenamiento de código y de estadísticas

Recurso	Coste
Code & Static Data Storage - First 1 GB	Gratuito
Code & Static Data Storage - Exceeding 1 GB	$0.13 por GB al mes

Tabla 7. Almacenamiento de código y estadísticas de Google App Engine

- Bases de datos

Recurso	Límite gratuito por defecto diario	Límite por defecto con *billing* activado
Stored Data	1 GB Nota: límite total, no diario.	1 GB gratis; sin máximo
Number of Indexes	200 Nota: límite total, no diario.	200
Write Operations	50,000	Ilimitado
Read Operations	50,000	Ilimitado
Small Operations	50,000	Ilimitado

Tabla 8. Configuraciones de bases de datos en Google App Engine

- Instancias disponibles

Recurso o llamada a la API	Cuota Gratuita
Frontend Instances (Automatic Scaling Modules)	28 free instance-hours per day
Backend Instances (Basic and Manual Scaling Modules)	8 free instance-hours per day

Tabla 9. Configuración por defecto de las instancias disponibles

- Google Search [18]

Recurso o llamada a la API	Cuota Gratuita
Total Storage (Documents and Indexes)	0.25 GB
Simple Queries	1,000 queries per day
Complex Queries	100 queries per day
Adding documents to Indexes	0.01 GB per day
Other API Calls (billed as operations)	1,000 operations per day

Tabla 10. Configuraciones disponibles para Google Search

Y los precios de los servicios más destacables [19].

Recurso	Unidad	Coste por unidad
Outgoing Bandwidth	Gigabytes	$0.12
Frontend Instances (F1 class)	Instance hours	$0.08
Frontend Instances (F2 class)	Instance hours	$0.16
Frontend Instances (F4 class)	Instance hours	$0.32
Frontend Instances (F4_1G class)	Instance hours	$0.48
Discounted Instances	Instance hours	$0.05
Backend Instances (B1 class)	Hourly per instance	$0.08
Backend Instances (B2 class)	Hourly per instance	$0.16
Backend Instances (B4 class)	Hourly per instance	$0.32
Backend Instances (B4_1G class)	Hourly per instance	$0.48
Backend Instances (B8 class)	Hourly per instance	$0.64
Stored Data (Blobstore)	Gigabytes per month	$0.13
Stored Data (Datastore)	Gigabytes per month	$0.18
Stored Data (Logs Data)	Gigabytes per month	$0.24
Stored Data (Task Queue)	Gigabytes per month	$0.24
Dedicated Memcache	Gigabytes per hour	$0.12
Channel	Channel opened	$0.0001 ($0.01/100 channels)
Recipients Emailed	Email	$0.0001
XMPP	XMPP stanzas	$0.000001 ($0.10/100,000 stanzas)
Logs API	Gigabytes	$0.12
SNI SSL certificates	Sets of five SNI certificate slots per month	$9.00
SSL Virtual IPs (VIPs)	Virtual IP per month	$39.00
PageSpeed bandwidth	Gigabytes (in addition to outgoing bandwidth charges)	$0.39

Tabla 11. Precios de los servicios disponibles en Google App Engine

Anexo II

Estudio de Arte: Amazon EC2

El desarrollador tiene la posibilidad de configurar las instancias contratadas de entre una multitud de variantes que *Amazon* pone a disposición, que pueden ir desde instancias de propósito general (configuración básica con 3.75 GiB de memoria y 4GB de almacenamiento local) hasta instancias de almacenamiento optimizado (con configuraciones de 117 GiB de memoria y hasta 24 x 2 TB de almacenamiento local) [20].

Tipo	Configuración	vCPU	Memoria	Unidades EC2	Almacenamiento SSD	Plataforma
General Purpose	m3.medium	1	3.75 GiB	3	4 GB	64-bit
General Purpose	m3.large	2	7.5 GiB	6.5	32 GB	64-bit
General Purpose	m3.xlarge	4	15 GiB	13	2 x 40 GB	64-bit
General Purpose	m3.2xlarge	8	30 GiB	26	2 x 80 GB	64-bit
Compute Optimized	c3.large	2	3.75 GiB	7	2 x 16 GB	64-bit
Compute Optimized	c3.xlarge	4	7 GiB	14	2 x 40 GB	64-bit
Compute Optimized	c3.2xlarge	8	15 GiB	28	2 x 80 GB	64-bit
Compute Optimized	c3.4xlarge	16	30 GiB	55	2 x 160 GB	64-bit
Compute Optimized	c3.8xlarge	32	60 GiB	108	2 x 320 GB	64-bit
GPU	g2.2xlarge	8	15 GiB	26	60 GB	64-bit
Memory Optimized	m2.xlarge	2	17.1 GiB	6.5	420 GB	64-bit
Memory Optimized	m2.2xlarge	4	34.2 GiB	13	850 GB	64-bit
Memory Optimized	m2.4xlarge	8	68.4 GiB	26	2 x 840 GB	64-bit
Memory Optimized	cr1.8xlarge	32	244 GiB	88	2 x 120 GB	64-bit
Storage Optimized	i2.xlarge	4	30.5 GiB	14	800 GB	64-bit
Storage Optimized	i2.2xlarge	8	61 GiB	27	2 x 800 GB	64-bit
Storage Optimized	i2.4xlarge	16	122 GiB	53	4 x 800 GB	64-bit
Storage Optimized	i2.8xlarge	32	244 GiB	104	8 x 800 GB	64-bit
Storage Optimized	hs1.8xlarge	16	177 GiB	35	24 x 2 TB	64-bit
Micro	t1.micro	1	613 MiB	hasta 2	solo EBS	32-bit o 64-bit

Tabla 12. Configuraciones disponibles en Amazon EC2

Por supuesto la capacidad de elección llega también a características tales como bases de datos (MongoDB, MySql, Microsoft SQL Server, etc), CRM (WordPress, Joomia!, Alfresco, etc) o incluso la pila de la aplicación (Tomcat, JBoss, Ruby on Rails, etc).

Algunos de los lenguajes más utilizados para el desarrollo sobre esta plataforma son JAVA, PHP, Python, Ruby, APL, WinDEv, etc.

Así mismo es posible configurar el sistema operativo a utilizar entre una amplia lista disponible:

- CentOS
- Amazon Linux
- Red Hat Enterprise Linux
- Debian
- Oracle Enterprise Linux
- Windows Server
- SUSE Enterprise Linux
- Ubuntu

Amazon EC2 contempla una división por regiones a las que dar servicio a nivel mundial, con ello se intenta mejorar problemas ocasionados por distancias tales como la latencia de las redes por ejemplo. Así las regiones que se contemplan son:

- US East (Northern Virginia)
- US West (Northern California)
- US West (Oregon)
- South America (Sao Paulo)
- EU West (Ireland)
- Asia Pacific (Singapore)
- Asia Pacific (Tokyo)
- Asia Pacific (Sydney)

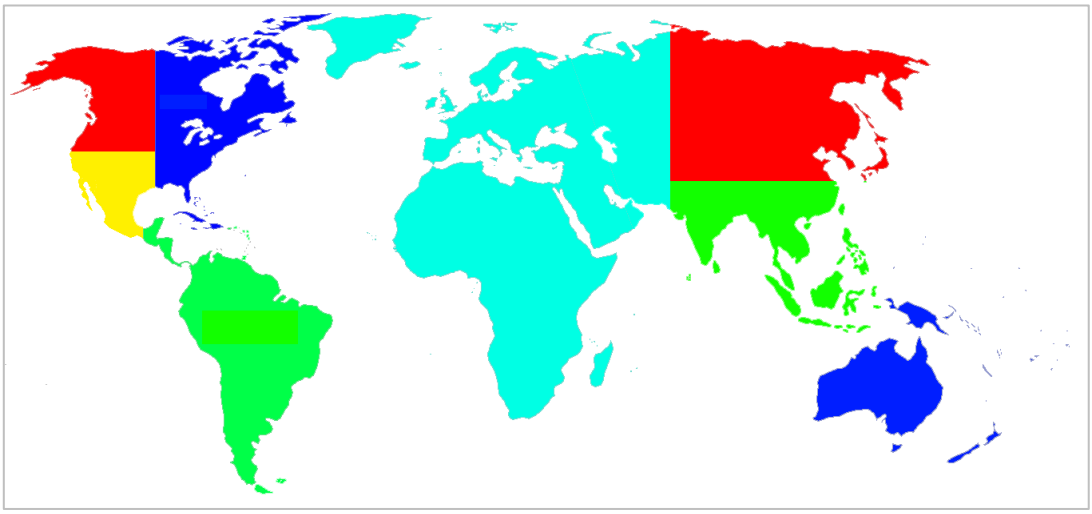


Figura 36. Distribución por zonas del servicio Amazon EC2

Los precios para los servicios disponibles se basan en este caso tanto en la zona geográfica en cuestión, en el tipo de configuración contratada como en el sistema operativo a utilizar. Por lo tanto, el catálogo de configuraciones disponibles basadas en Linux para la región de UE West (Irlanda) sería [21].

Tipo	Configuración	Precio por uso
General Purpose	m3.medium	$0.124 per Hour
General Purpose	m3.large	$0.248 per Hour
General Purpose	m3.xlarge	$0.495 per Hour
General Purpose	m3.2xlarge	$0.990 per Hour
Compute Optimized	c3.large	$0.171 per Hour
Compute Optimized	c3.xlarge	$0.342 per Hour
Compute Optimized	c3.2xlarge	$0.683 per Hour
Compute Optimized	c3.4xlarge	$1.366 per Hour
Compute Optimized	c3.8xlarge	$2.732 per Hour
GPU	g2.2xlarge	$0.702 per Hour
Memory Optimized	m2.xlarge	$0.460 per Hour
Memory Optimized	m2.2xlarge	$0.920 per Hour
Memory Optimized	m2.4xlarge	$1.840 per Hour
Memory Optimized	cr1.8xlarge	$3.750 per Hour
Storage Optimized	i2.xlarge	$0.938 per Hour
Storage Optimized	i2.2xlarge	$1.876 per Hour
Storage Optimized	i2.4xlarge	$3.751 per Hour
Storage Optimized	i2.8xlarge	$7.502 per Hour
Storage Optimized	hs1.8xlarge	$4.900 per Hour
Micro	t1.micro	$0.020 per Hour

Tabla 13. Precios de los servicios de Amazon EC2

Anexo III

Estudio de Arte: JDO vs JPA

Una comparativa funcional entre las dos especificaciones sería [22]:

Funcionalidad	JDO	JPA
JDK mínimo	1.3+	1.5+
Especificaciones Java	J2EE, J2SE	J2EE, J2SE
Mecanismos de persistencia	XML, Anotaciones, API	XML, Anotaciones
Datastore soportados	Any	RDBMS (Bases de datos relacionales)
Soporte a campos 'transient'	Sí	No
Transacciones	Pessimistic, Optimistic	Optimistic, some locking
Object Identity	datastore-identity, application-identity	application-identity
Object Identity generation	Sequence, Table, Identity, Auto, UUID String, UUID Hex	Sequence, Table, Identity, Auto
Tipo de objetos soportados	Java primitive types, wrappers of primitive types, java.lang.String, java.lang.Number, java.math.BigInteger, java.math.BigDecimal, java.util.Currency, java.util.Locale, java.util.Date, java.sql.Time, java.sql.Date, java.sql.Timestamp, java.io.Serializable, boolean[], byte[], char[], double[], float[], int[], long[], short[], java.lang.Object, interface, Boolean[], Byte[], Character[], Double[], Float[], Integer[], Long[], Short[], BigDecimal[], BigInteger[], String[],PersistenceCapable[], interface[], Object[], Enums, java.util.Collection, java.util.Set, java.util.List, java.util.Map, Collection/List/Map of simple types	Java primitive types, wrappers of the primitive types, java.lang.String, java.math.BigInteger, java.math.BigDecimal, java.util.Date, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.Serializable, byte[], Byte[], char[], Character[], Enums, java.util.Collection, java.util.Set, java.util.List, java.util.Map Collection/List/Map of persistable types
Objetos embebidos	Embedded persistent objects, Embedded Collections, Embedded Maps	Embedded persistent objects
Herencia	Cada clase tiene su propia estrategia	La clase Root define la estrategia

Configuración de operaciones en cascada	delete	persist, delete, refresh
Query Language	JDOQL, SQL, others	JPQL, SQL
Query language case sensitivity	JDOQL lowercase/UPPERCASE	JPQL case-insensitive
Bulk update/delete	JDOQL Bulk Delete	JPQL Bulk Delete, JPQL Bulk Update
RDBMS Schema Control	Tables, columns, PK columns, PK constraints, FK columns, FK constraints, index columns, index constraints, unique key columns, unique key constraints	Tables, columns, PK columns, FK columns, unique key columns
ORM Relationships	Full range of Collection, Map, List, Array, 1-1, 1-N, M-N using PC, Non-PC and interface objects	Basic 1-1, 1-N, M-N, Collection<NonPC>, Map<NonPC>
Default ORM column size	256	255
Default ORM identifiers (tables/columns)	No	Yes
Default ORM mappings	Yes, JDBC types defined for Java types	No
Caching interface	L2 Caching API	L2 Caching API

Tabla 14. Comparativa de servicios entre JPA y JDO

Se puede considerar por lo tanto a *JPA* como un subconjunto de funcionalidades de *JDO*.

Anexo IV

Estudio de Arte: GIT

Teniendo en cuenta estos aspectos, el manejo de este sistema de control se basa en las siguientes funcionalidades con las que aprovechar todo el potencial de *GIT* [23]:

Comando	Funcionalidad
git add	Añadir cambios o ficheros al *Staging Area*
git branch	Listar o crear ramas de desarrollo
git clone	Clonar un repositorio a un directorio
git commit	Publicar los cambios al repositorio local
git diff	Mostrar las diferencias entre versiones
git init	Crear un repositorio vacío
git log	Mostrar el histórico de "*commit*" realizados
git merge	Unir dos o más ramas de desarrollo
git mv	Mover o renombrar un directorio
git pull	Recoger los ficheros de otro repositorio
git push	Enviar los ficheros al repositorio remoto
git reset	Deshacer los cambios y volver al estado anterior
git rm	Eliminar ficheros del árbol de desarrollo
git status	Mostrar el estado de la rama de trabajo actual

Tabla 15. Listado de los comandos básicos en GIT

Hay que destacar también que el diseño de *GIT* se basaba en el nivel más bajo del servicio proporcionando soporte a otros clientes tales como *Cogito*, *StGIT o GITHUB*. Aunque actualmente ya es capaz de proporcionar un servicio completo.

Anexo V

Estudio del Arte: Elección final de componentes

V. 1. Comunicación cliente - servidor: Google Cloud EndPoints

Tras el estudio de todos los componentes, éste ha sido el que ha cobrado mayor importancia debido a su relevancia en el desarrollo global de la aplicación. Como se ha comentado, todos los componentes pueden ser técnicamente conectados entre sí y a su vez la lógica propia de cada uno de ellos tratada de forma independiente.

Pero en un entorno distribuido, marcado sobre todo por el uso de *La Nube,* el modo de integración y conexión entre componentes puede hacer que el desarrollo de la aplicación se simplifique de forma considerada, lo que a su vez repercutirá en el coste del desarrollo y mantenimiento, el tiempo necesario para completar el proyecto, la configuración y especialización del equipo e incluso en el mercado de usuarios para los que la aplicación estará concebida.

Siguiendo estas pautas, el servicio ofrecido por *Google* conocido como *Google Cloud EndPoints* se presenta como la solución ideal. Caracterizado por enmascarar casi toda la lógica necesaria para conectar un servidor de aplicaciones con sus clientes móviles, el uso de esta tecnología convierte en casi transparente muchos de los puntos más críticos para el desarrollador, los relacionados con la comunicación entre distintos dispositivos.

Pero además no solo implementa de una forma sencilla y amigable el acceso desde el cliente móvil, sino que reduce a básicamente un solo click la generación en el servidor de la lógica necesaria para permitir una comunicación externa.

Desde el punto de vista del cliente móvil esta tecnología resuelve muchos de los problemas que una comunicación remota presenta, como el establecimiento y mantenimiento de la conexión, la compatibilidad y parseo de la información tanto enviada como recibida o la configuración de la seguridad empleada en la comunicación.

El servicio que presta está preparado para ser consumido principalmente por las grandes plataformas móviles que en la actualidad están marcando el mercado, como son *Android, HTML5 y iOS*. Además por extensión y gracias a las compatibilidades entre sistemas, es posible hacer uso de este servicio desde otras plataformas como *BlackBerry o Firefox OS*. E incluso, debido a la naturaleza básica de la tecnología de comunicación que usa (*RESTFul*) sería posible acceder desde el resto de sistemas aún no contemplados, como podría ser *Windows 8*.

Parece claro que apostar por esta tecnología puede marcar enormemente el desarrollo global de la aplicación y del sistema en su conjunto completo.

V. 2. Servidor de aplicaciones: *Google App Engine*

Con los datos recogidos para ambas plataformas, *Google App Engine y Amazon EC2*, es posible realizar una comparativa más objetiva desde el punto de vista técnico. Hay que tener en cuenta que *Amazon EC2* es considerado popularmente como un modelo *IaaS* mientras que *Google App Engine* es claramente un modelo *PaaS*, con lo que el primero en sí provee de mayor número de funcionalidades y configuraciones que el segundo pero a su vez eso mismo incrementa la complejidad de configuración y mantenimiento de la infraestructura [24].

- Aspectos generales

	Google App Engine	Amazon EC2
Coste del plan básico	$0 per hour	$0.011 per hour
Categoría del modelo	Infrastructure as a Service	Platform as a Service
Licencia del software	Propietaria	Propietaria
Año de fundación	2002	2006
Interfaz de control	Web Base Application API Línea de comandos Graphical User Interface	Web Base Application API

Tabla 16. Comparativa general entre Google App Engine y Amazon EC2

- Planes

	Google App Engine	Amazon EC2
Detalles del plan básico	500 MB gratuitos con ancho de banda y CPU suficiente hasta para 5 millones de páginas visitadas al mes	1.7GB RAM 160 GB almacenamiento interno 1 EC2 Unit
Tipo de suscripción	Pago por consumo	Pago por consumo o planes de suscripción

Tabla 17. Comparativa de los planes ofrecidos por Google App Engine y Amazon EC2

- Funcionalidades disponibles

	Google App Engine	Amazon EC2
Seguridad en modo gratuito	Persistencia	Advanced Firewall Privacidad de los datos críticos Permisos básicos o customizados Recuperación ante fallos
Seguridad en modo de pago	Almacenamiento de backups	Advanced Firewall Privacidad de datos críticos Encriptación de datos Detección de intrusos al sistema Persistencia de datos Snapshot backup
Autoescalabilidad	Sí, gratuito	Sí, gratuito
Virtual Private Servers	-	Sí, bajo cargo extra
Root Access	Sí	Sí
Guarantee Network Availability	99,9%	99,999%
Balanceo de carga	Sí, gratuito	Sí, bajo cargo extra
Monitorización	-	Sí, gratuito
Servicio de almacenamiento de ficheros	Sí, bajo cargo extra	Sí, bajo cargo extra
Sistemas Operativos	Linux	CentOS Debian Fedora Gentoo Linux Solaris SUSE Linux Oracle Enterprise Linux Red Hat Enterprise Linux
Lenguajes de programación	JAVA Python PHP Go	APL JAVA PHP Python Ruby WinDev
Servicios de Soporte en caso de problemas	Foro Online	Teléfono Foro 24/7 Respuesta urgente Herramientas de diagnóstico Guías online

Tabla 18. Comparativa de las funciones disponibles en Google App Engine y Amazon EC2

Las prestaciones y funcionalidades ofrecidas por ambos modelos son de gran similitud y en el sector del desarrollo sobre *Cloud Computing* hay multitud de reflexiones que eligen a uno y otro sistema como el más apropiado para cada caso. Un sistema con mayor necesidad de configuración para servicios más específicos podría decantarse por *Amazon EC2* ya que cubre mayor rango de funcionalidades al tratarse de un proprio *IaaS*.

Los sistemas desarrollados en *Google App Engine* gozan a su vez de mayor integración con la multitud de servicios que la misma plataforma de *Google* ofrece. Servicios de muy alta calidad y de una gran diversidad de funcionalidades que rápidamente incrementa el propio valor de las aplicaciones.

Es por esto que esta última característica es la base para la elección de *Google App Engine* como el servidor de aplicaciones que se utilizará en este proyecto. Es de gran importancia la integración de este servicio con la tecnología *Google Cloud EndPoints* que se ha definido como la base principal del proyecto. Igualmente es de valorar la integración que *Google* ofrece con algunos de los *IDEs* a utilizar lo que simplifica enormemente el trabajo del desarrollador.

V. 3. Bases de datos: *Object persistence*, *JDO*

Una vez se ha definido a *Google App Engine* como la plataforma sobre la que se ejecutará la lógica de la aplicación y a la tecnología *Google Cloud EndPoints* como el modo más óptimo para la comunicación cliente - servidor, el resto de componentes a utilizar serán elegidos teniendo en cuenta su afinidad con esos componentes y la facilidad de integración y de uso en el momento de desarrollar la aplicación final.

Este es el caso de la base de datos. Siguiendo las especificaciones que *Google Cloud EndPoints* marca para su desarrollo, los sistema de persistencia de objetos (*JDO y JPA*) son los sistemas que mejor encajan en este diseño. El uso de este modelo encaja perfectamente con las funcionalidades ofrecidas y las pautas requeridas por *Google Cloud EndPoints* lo que ayuda a un desarrollo más rápido, integrado y eficiente.

Y como se ha visto en la descripción de este modelo, la especificación *JPA* se puede considerar como un subconjunto de funcionalidades de *JDO* y dado que *Google App Engine* hace uso por igual de ambas especificaciones se ha decidido que *JDO* será el modelo de bases de datos a utilizar en este proyecto.

V. 4. Lenguajes de programación

Otra elección a tener en cuenta son los lenguajes de programación de los que se harán uso a lo largo del desarrollo de la aplicación. Estos serán determinados por las plataformas a usar y los clientes sobre los que se desarrollará la aplicación final.

Por lo cual los lenguajes a utilizar serán:

- *JAVA*: Para el desarrollo de la lógica de negocio en el servidor de aplicaciones *Google App Engine*. Aunque como se ha visto este servicio soporta multitud de lenguajes de programación, *JAVA* será utilizado por la familiaridad para el equipo de trabajo.

- *Android*: Haciendo uso de su *SDK* en el desarrollo del cliente que correrá sobre el sistema operativo con el mismo nombre. Igualmente y gracias a la compatibilidad entre plataformas, *BlackBerry* ofrece la posibilidad de usar este mismo lenguaje para el desarrollo de la aplicación que correrá sobre su sistema operativo.

- *HTML5*: Será el lenguaje de programación base para los clientes *WEB* que accederán a la lógica de negocio final.

V. 5. Entorno de desarrollo integrado: *Eclipse*

En el desarrollo sobre la plataforma *Google App Engine*, el uso de *Eclipse* o de *IntelliJ* es recomendado de igual forma. Pero la cantidad de funciones integradas en *Eclipse* mediante los *plugins* ofrecidos por *Google* y la familiaridad en el uso con este *IDE* hace que sea *Eclipse* el entorno perfecto para el grupo de trabajo de este proyecto.

V. 6. Sistema de control de versiones: *GITHUB*

Son multitud de funcionalidades las que *GIT* ha introducido como evolución a los otros sistemas evaluados, *CVS y SVN*. El entorno distribuido del que se compone este equipo de trabajo y las características propias de este sistema lo convierten en el modelo más adecuado para este proyecto.

GITHUB además se presenta como la implementación más popular que en la actualidad es posible encontrar dentro de un marco *open source* de acceso gratuito.

Anexo VI

Configuración y preparación de herramientas

En este capítulo se detallan las herramientas que han sido necesarias para la elaboración de este proyecto así como una breve descripción del proceso de configuración de las mismas.

La lista de herramientas utilizadas es la siguiente:

- JAVA
- Eclipse
- Google Plugin for Eclipse
- Android SDK
- Android Developer Tool for Eclipse
- Terminal real Android para pruebas
- Plugin de *GITHUB* para *MAC*
- *Firefox* y *Firebug Tool* para *Firefox*
- *RESTClient*
- *Google App Engine Tool*
- *Google API Explorer*
- *Google Drive*

VI. 1. *JAVA*

La actual versión estable disponible en el mercado es *JAVA 7. Oracle* es la empresa encargado del desarrollo de este lenguaje y es la distribuidora de sus diferentes versiones y paquetes [25].

Para el desarrollo de este proyecto de fin de carrera es necesaria la distribución estándar del mismo *J2SE* y es recomendable la instalación de su paquete *JDK*. El proceso de instalación consiste en seguir los pasos proporcionados por el instalador descargado.

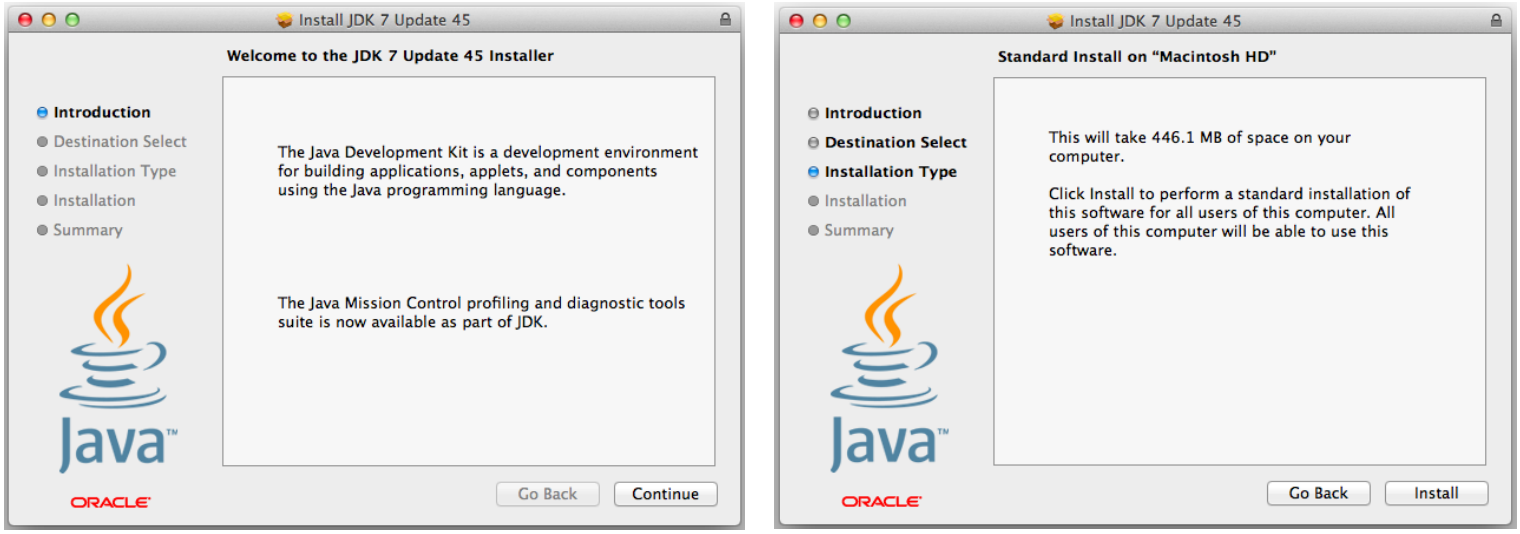


Figura 37. Proceso de instalación de Java

El componente va indicando en cada momento el estado de la instalación así como información útil de los requerimientos para completar dicha instalación.

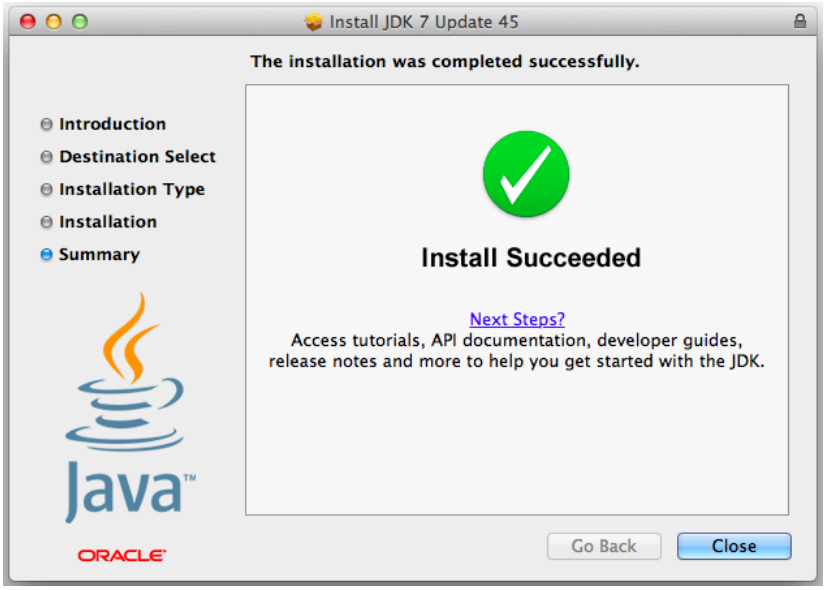


Figura 38. Instalación de Java completada

Es importante saber después de terminar este proceso cuál es la ruta en la que el nuevo *JDK* ha sido instalado. Esta información será necesaria para la configuración de futuros componentes necesarios para el proyecto. Para ello se puede ejecutar en cualquier ventana de comando o terminal el siguiente comando:

```
$ /usr/libexec/java_home -v 1.7
/Library/Java/JavaVirtualMachines/jdk1.7.0_45.jdk/Contents/Home
```

Código 55. Localización de la ruta de instalación de JAVA

Esta es pues la ruta de nuestro *JAVA 7 JDK* instalado.

VI. 2. *Eclipse*

Este *IDE open source* puede ser descargado de forma gratuita directamente desde su página web [26]. Su versión estándar nos ofrece todas las herramientas que podemos necesitar de un *IDE* de este tipo.

Luna o versión 4.4.1 es la última versión estable disponible y la usada para este proyecto. Su distribución *Mac OS X (Cocoa 64)* es la idónea para nuestro entorno de trabajo.

En este caso no se trata de una instalación en el sistema. El programa distribuido necesitará únicamente de una descompresión del fichero descargado en la ruta en la que deseemos configurar nuestro entorno.

En la primera ejecución del mismo deberemos especificar la ruta de nuestro *workspace* donde los proyectos serán creados por defecto.

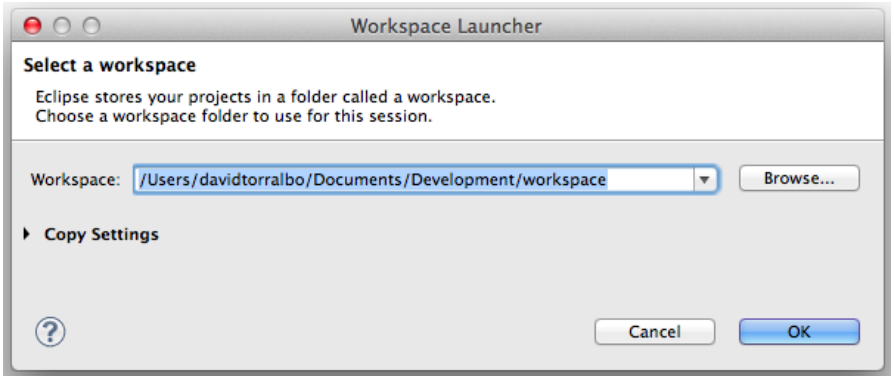


Figura 39. Pantalla de selección del workspace a utilizar en Eclipse

En esta ruta se crearan los proyectos junto a su estructura de ficheros y carpetas necesaria. Esta ruta podrá ser modificada individualmente a la hora de crear cada nuevo proyecto.

La instalación inicial carece de configuración *JAVA*, por lo que deberemos especificar la ruta de nuestro JAVA JDK instalado. Para ello se accede a las propiedades de Eclipse (Eclipse -> Preferences) y seleccionamos la configuración *Installed JREs* bajo la entrada *JAVA*:

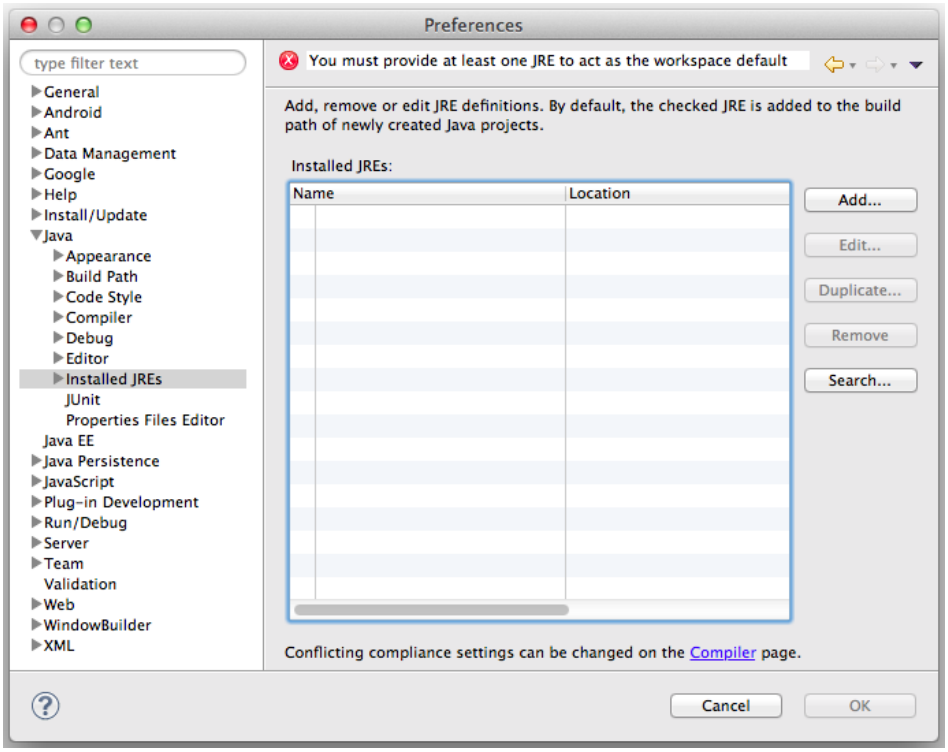


Figura 40. Configuración de la ruta de Java en Eclipse

Cuando seleccionamos el botón *Add* para añadir nuestro *JRE* disponible, debemos especificar qué tipo de *JRE* vamos a configurar. Para este proyecto necesitaremos un *JRE* de tipo *Standard VM*.

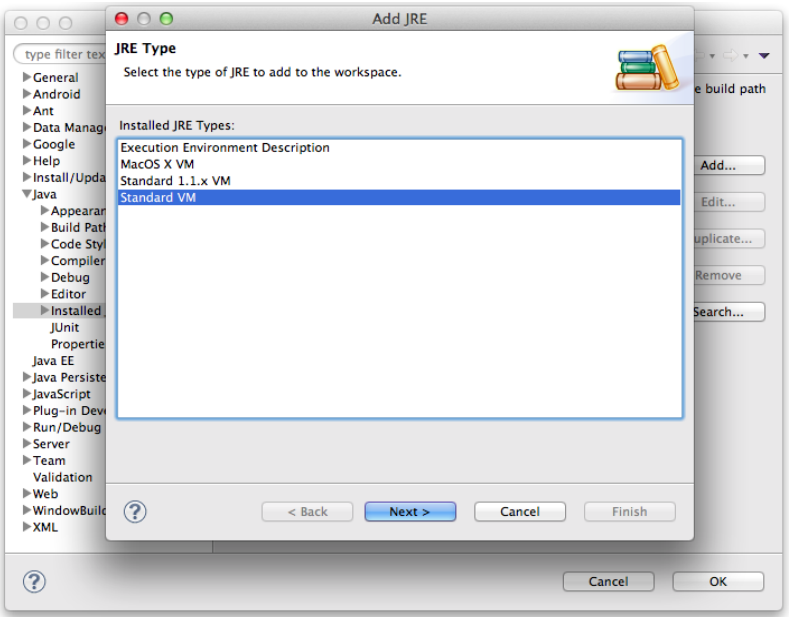


Figura 41. Selección del tipo de JRE a configurar

Se debe especificar la ruta en la que se encuentra la máquina virtual, obtenida en el apartado anterior sobre la instalación de *JAVA*.

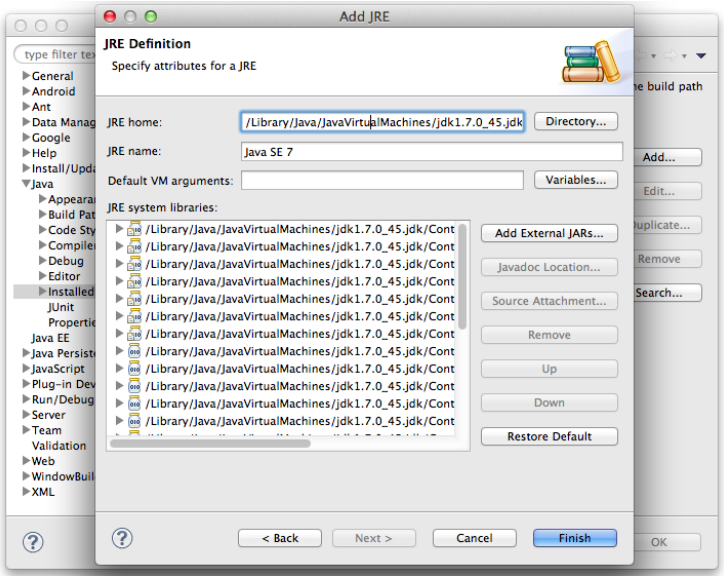


Figura 42. Configuración de la ruta de Java

Podemos modificar el nombre del *JRE* para hacerlo más legible y reconocible en todo momento.

Una vez configurado este apartado, el entorno de trabajo o *IDE* se encuentra disponible para poder crear y gestionar proyectos basados en *JAVA*.

VI. 3. *Google Plugin for Eclipse*

Para un desarrollo más integrado y cómodo sobre los servicios de Google, como el servidor *Google App Engine* o la generación de los *Google Cloud Endpoints*, es conveniente la instalación del *plugin* que Google proporciona específicamente para Eclipse.

Para la configuración de este *plugin* hacemos uso de la herramienta de instalación de nuevo *software* que *Eclipse* proporciona en *Help -> Install New Software…*

Dependiendo de la versión de Eclipse que se use, es este caso *Kepler* o 4.3, deberemos usar diferentes direcciones de dónde obtener el plugin. Para ver la relación entre la versión de *Eclipse* y la dirección del *plugin* se puede consultar la documentación *online* disponible por Google.

Así, para *Kepler* o 4.3 se debe usar la siguiente dirección:

- *https://dl.google.com/eclipse/plugin/4.3*

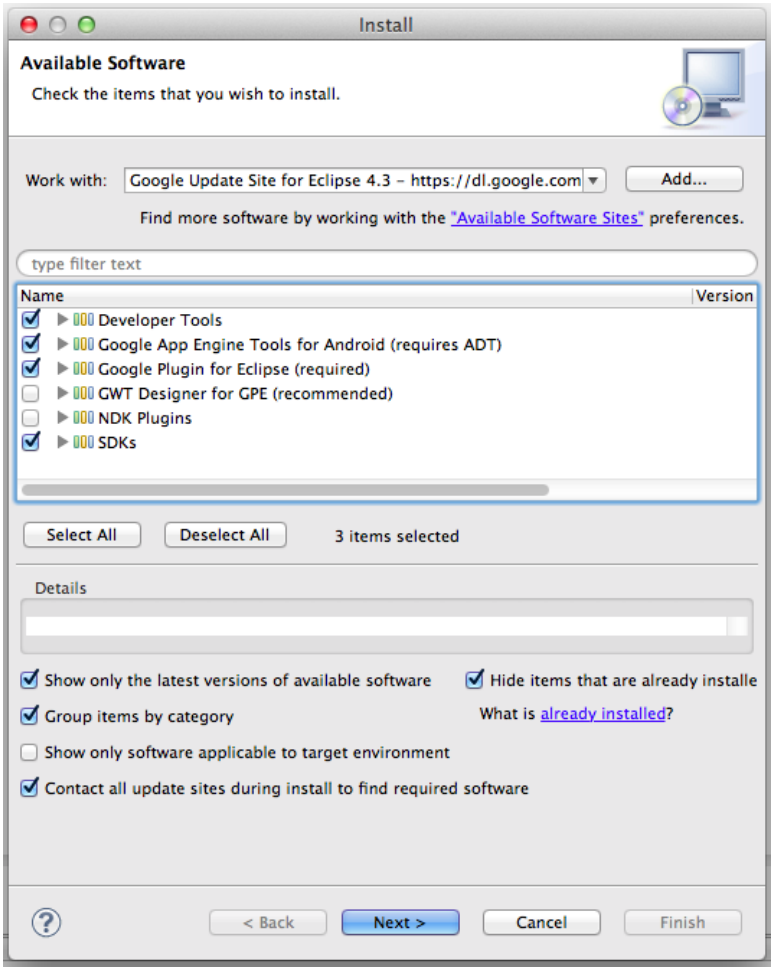


Figura 43. Instalación del plugin de Google para Eclipse

El servicio mínimo necesario que debe ser instalados para este proyecto es *Google Plugin for Eclipse*, pero también es recomendable instalar otros ofrecidos desde el mismo servidor que nos ayudarán a una mejor integración con los servicios de *Google*, por ejemplo, para el enlace entre el servidor en *Google App Engine* y el cliente en *Android*. Así pues se instalan los siguientes servicios:

- *Developer Tool*
- *Google App Engine Tool for Android*
- *Google Plugin for Eclipse*
- *SDKs*

Se deben confirmar los paquetes a instalar así como aceptar los términos y condiciones expuestos por Google.

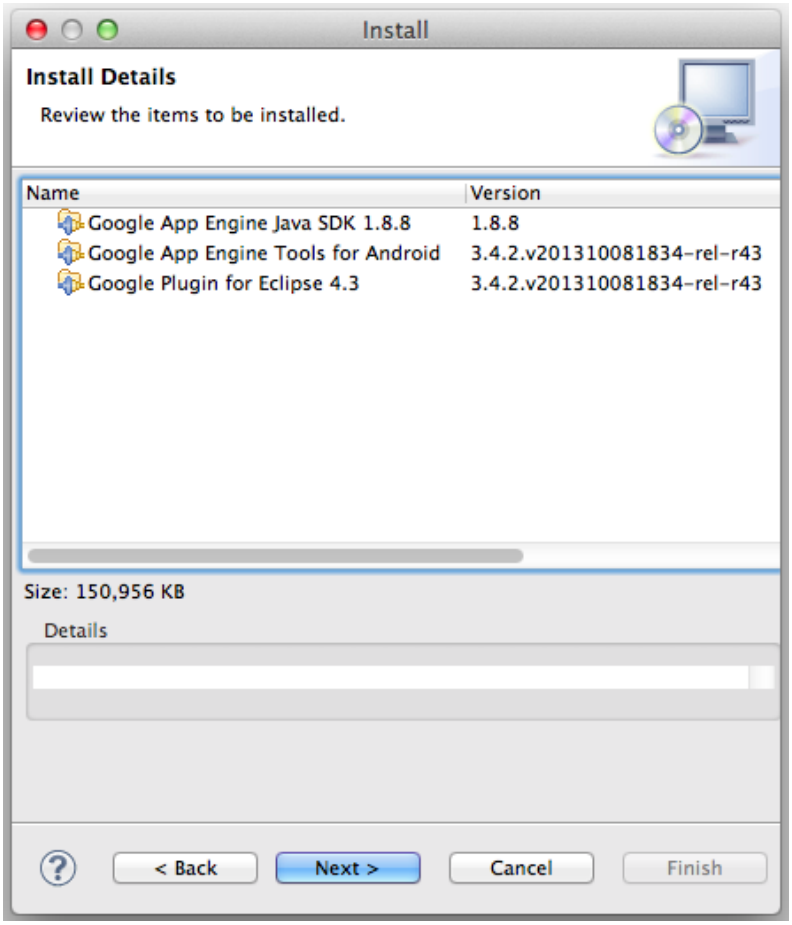


Figura 44. Selección de los paquetes a instalar

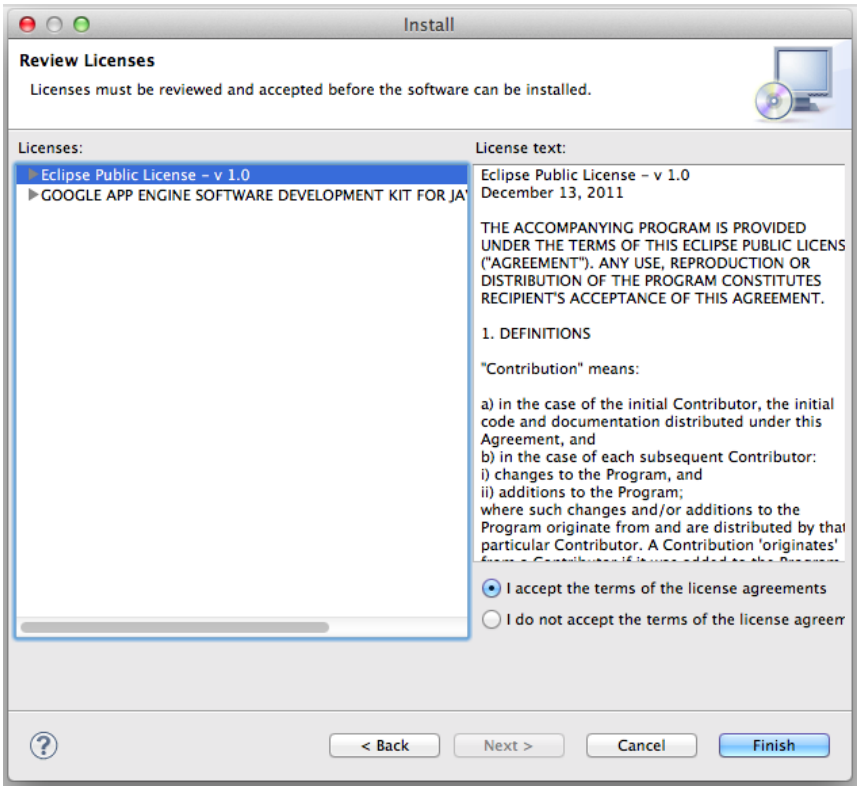


Figura 45. Confirmación de la instalación

Y una vez terminado el proceso de instalación ya tendremos el *Plugin de Google* para *Eclipse* con el que poder utilizar las herramientas que *Google* ofrece para los desarrolladores.

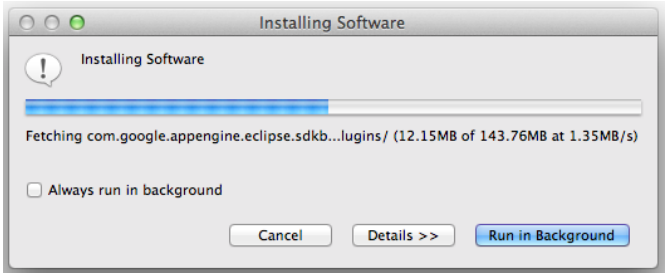


Figura 46. Barra de progreso de la instalación del plugin de Google para Eclipse

Una vez instalados los paquetes, se verán reflejados los cambios una vez se haga un reinicio *Eclipse*. Se puede ver entonces como nuevas funcionalidades han sido añadidas a la herramienta de trabajo del entorno.

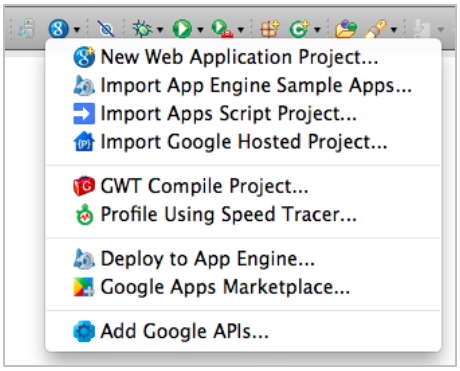


Figura 47. Nuevas funcionalidades disponibles gracias al plugin de Google

Se puede comprobar también la instalación más importante que este *plugin* proporciona, el propio SDK de Google App Engine, en este caso en su versión 1.8.8. Su configuración automática se puede ver en las preferencias de *Eclipse* (Eclipse -> Preferences), en la sección de *Google -> App Engine*.

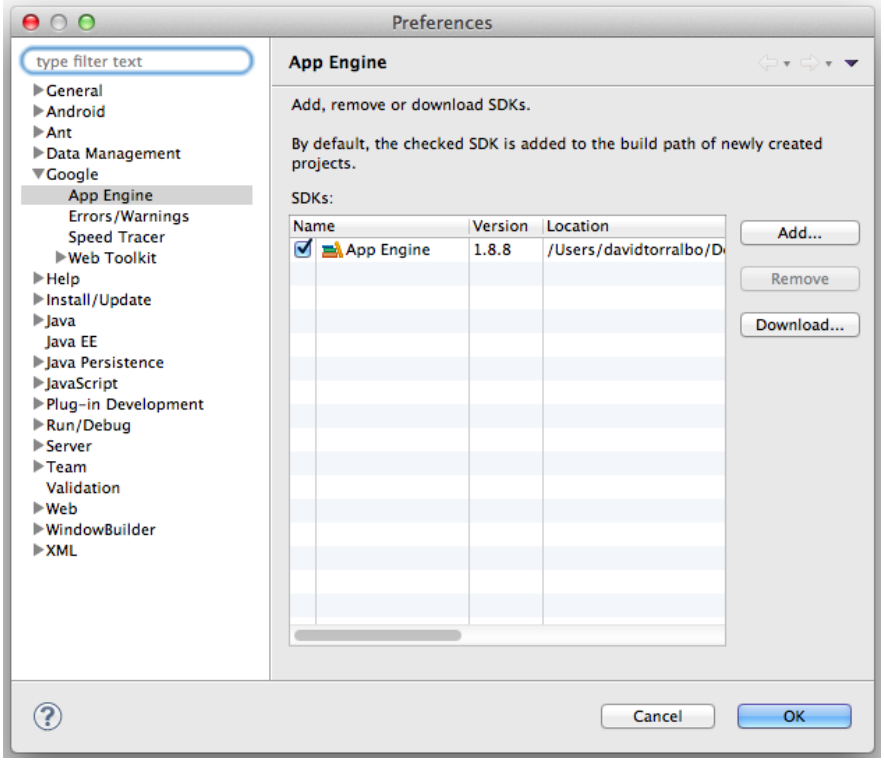


Figura 48. Configuración de App Engine en Eclipse

VI. 4. Android *SDK*

La instalación del *SDK* de *Android* puede realizarse por diferentes caminos paralelos. Los recomendados en la documentación de *Google* son a través del *bundle* que proporcionan, en el que se agrupan el SDK necesario junto con el *Plugin ADT* (*Android Developer Tool*), o mediante la descarga individual del *SDK* de *Android*.

Para este desarrollo se decidió descargar por separado el *SDK* y el *Plugin* para una más detallada explicación de sus instalaciones. Así la versión más acorde al entorno de trabajo es *Mac OS X 32 & 64-bit*. Una vez descargado el paquete, su descompresión genera la siguiente estructura de directorios, la cual ya es válida para su uso y configuración dentro del entorno de Eclipse.

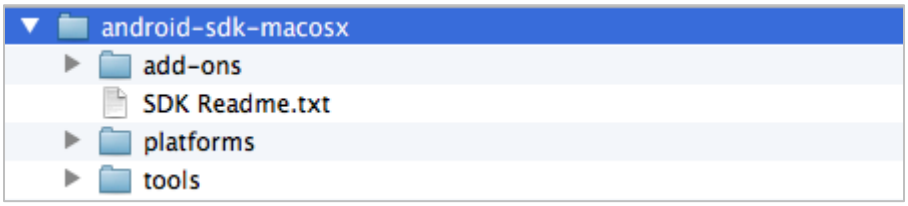


Figura 49. Estructura de directorios del Android SDK

Una vez instalado el *SDK* para *Android* es conveniente configurar su ruta dentro de las preferencias de Eclipse. Para ellos solo hay que especificar dicha ruta en *Eclipse -> Preferences* y ahí, dentro de la configuración para *Android*.

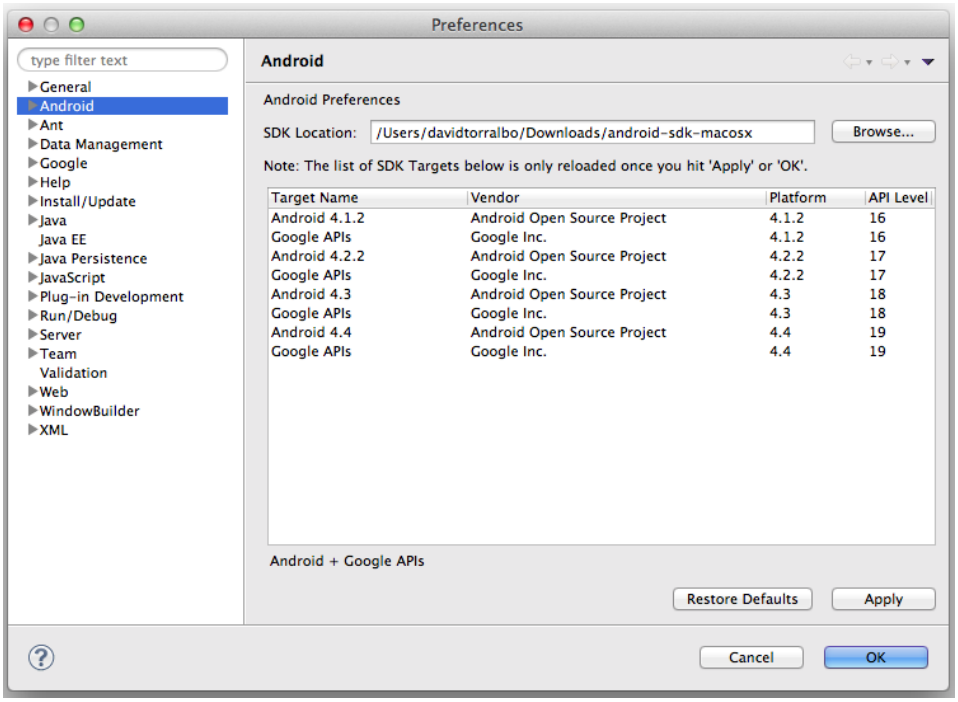


Figura 50. Configuración del Android SDK en Eclipse

VI. 5. *Android Developer Tool for Eclipse*

Con este *plugin* se tendrán disponibles multitud de servicios que son de gran ayuda a la hora del desarrollo de aplicaciones *Android*. Para su configuración, siguiendo los pasos indicados por la documentación *online* de *Google*, debemos instalarlo a través de la herramienta de instalación de nuevo *software* (*Help -> Install New Software...*).

La dirección desde donde este *plugin* puede ser descargado es:

- *https://dl-ssl.google.com/android/eclipse/*

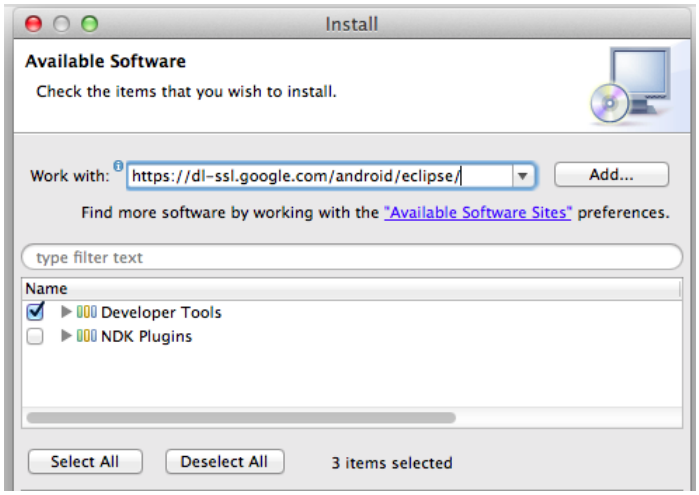


Figura 51. Instalación de ADT para Eclipse

Se debe confirmar y aceptar los términos y condiciones expuestos por *Google* para poder completar la instalación de este *Plugin*.

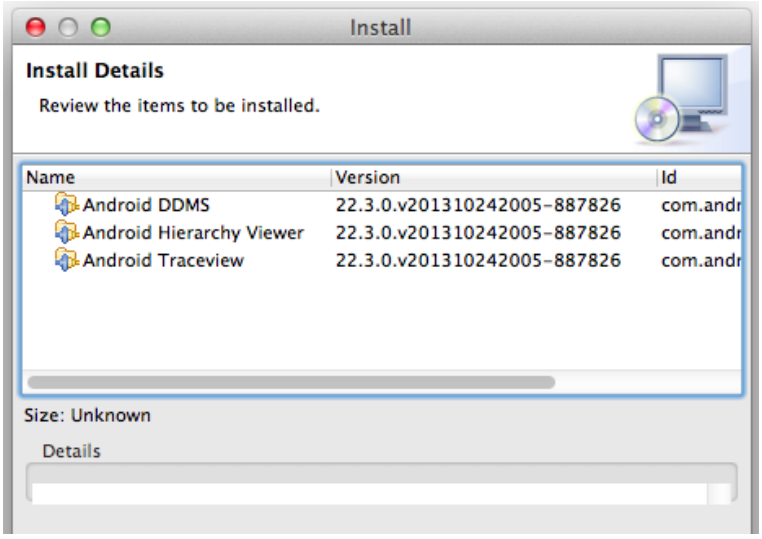


Figura 52. Confirmación de los paquetes a instalar

Una vez reiniciado *Eclipse* se puede encontrar las nuevas funcionalidades disponibles para el desarrollo de aplicaciones *Android*.

Figura 53. Nuevas funcionalidades disponibles gracias al ADT para Eclipse

Una vez se haya completado esta configuración, se puede acceder al *Android SDK Manager* que será de ayuda para descargarse las herramientas necesarias para el desarrollo de aplicaciones Android. Seleccionando el botón ahora presente en la barra de herramientas de Eclipse lanzamos este asistente.

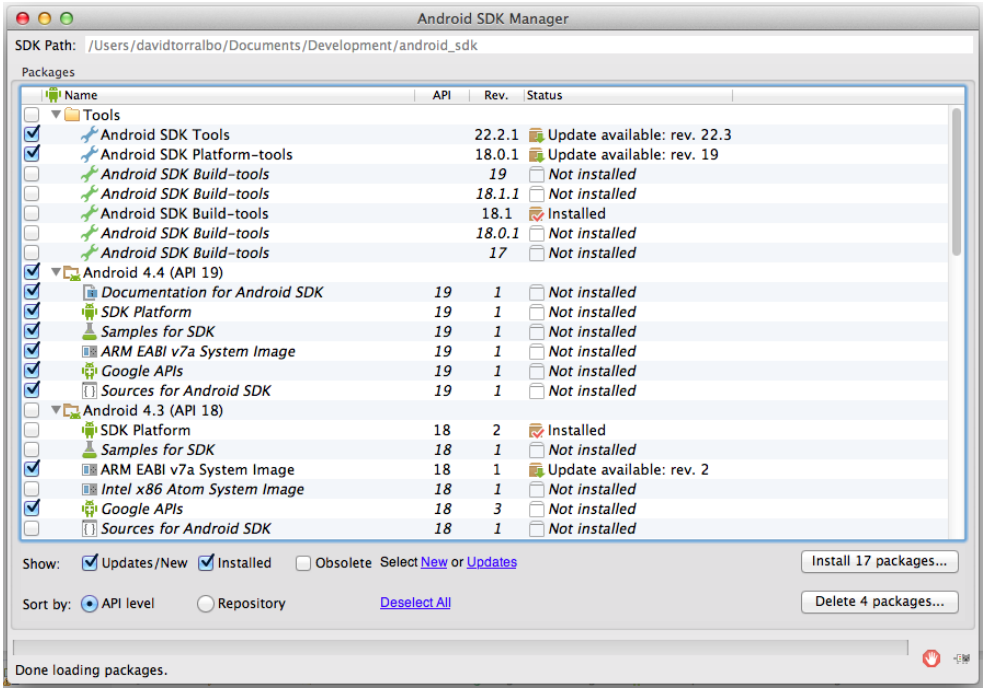


Figura 54. Panel del Android SDK Manager

En este panel es posible ver las diferentes versiones del SDK de Android hasta el momento disponible, y desde aquí es posible instalar los diferentes paquetes que cada versión contiene. Es recomendable elegir los paquetes más adecuados al desarrollo que en cada caso se va a realizar.

Es conveniente instalar el *SDK* y las *APIs* de *Google* de las versiones a las se quiera soportar con la aplicación Android. Se debe confirmar y aceptar los términos y condiciones expuestos por *Google*.

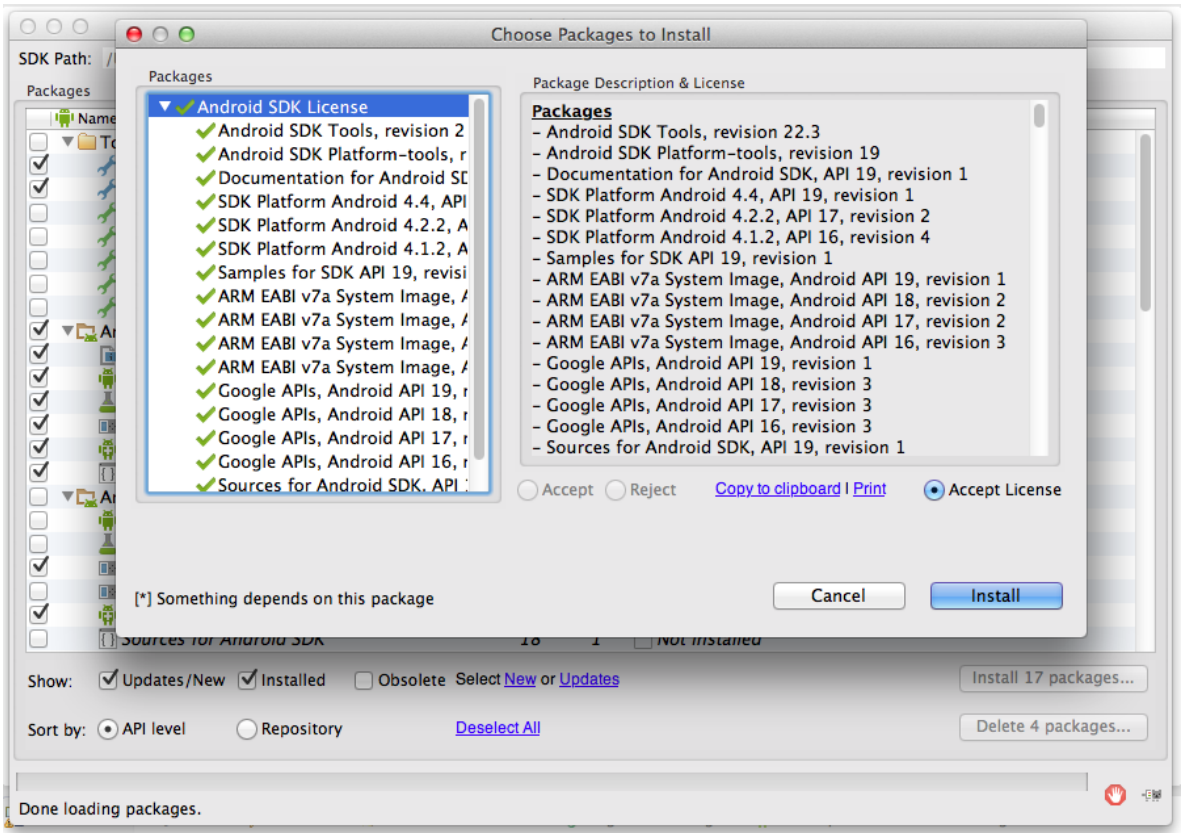


Figura 55. Listado de paquetes a instalar

Una vez instalado y configurado el *Android SDK Manager* se puede configurar igualmente el *Android Virtual Device Manager* (o *AVD*) con el que podremos simular el comportamiento de diferentes dispositivos *Android* así como diferentes versiones del sistema operativo.

Figura 56. Android Device Manager en Eclipse

En este caso, podremos configurar simulaciones de terminales reales cargados en el propio *SDK* instalado, como pueden ser la familia *Nexus*, o crear configuraciones propias con los parámetros que se puedan necesitar.

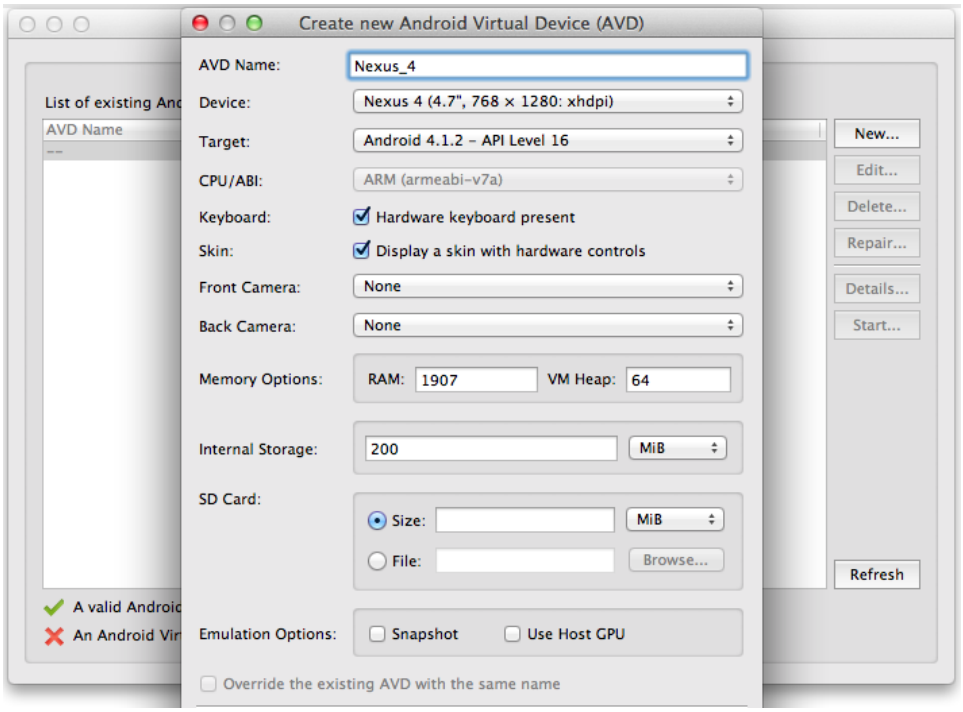


Figura 57. Configuración de un nuevo dispositivo en el Android Device Manager

Una vez configurado, el nuevo dispositivo virtual está disponible para su elección dentro de la lista de dispositivos en el momento de probar una aplicación.

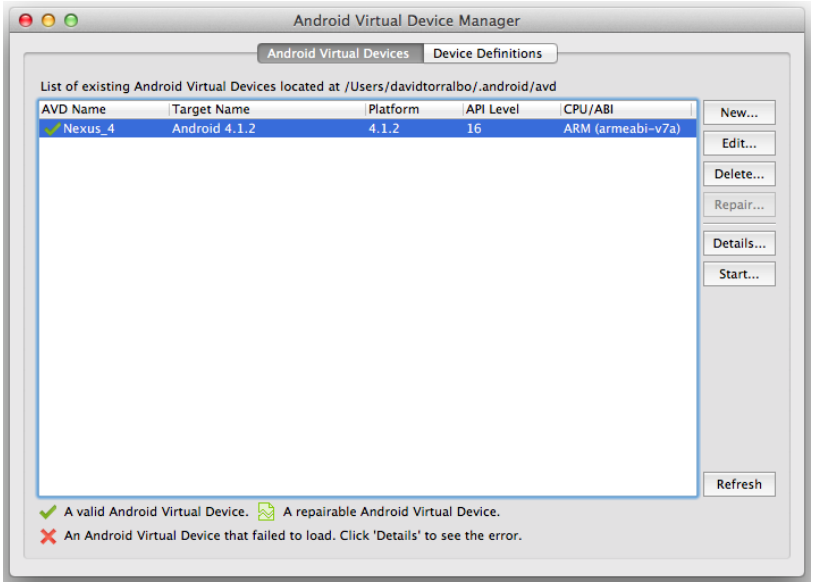


Figura 58. Listado de dispositivos virtuales disponible en el Android Device Manager

Para ver el funcionamiento del dispositivo virtual creado solo es necesario seleccionar el botón *Start*.

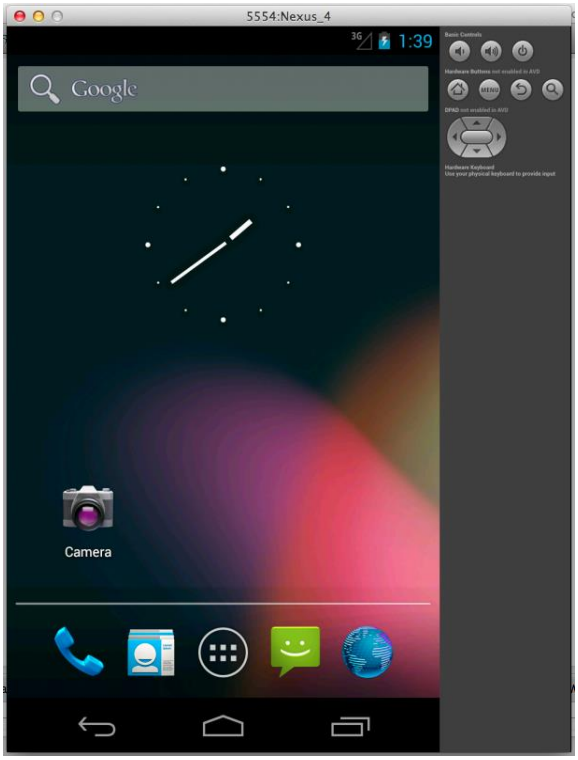


Figura 59. Simulación de un dispositivo virtual

Este componente permite probar y depurar el código que deseemos ejecutar en un terminal *Android*. Se encuentra vinculado con *Eclipse* por lo que es posible realizar depuración en tiempo real, ejecutar puntos de ruptura y ver con mayor nivel de detalle el comportamiento del código desarrollado.

Este simulador se convierte en una herramienta de gran utilidad a la hora de desarrollar aplicaciones *Android*, gracias al cual es posible ver el comportamiento y la renderización de la aplicación en distintas configuraciones tanto de memoria, CPU y de formato de pantalla del terminal.

VI. 6. Terminal real Android para pruebas

También es posible usar un terminal real con sistema operativo *Android* como dispositivo de pruebas y depuración mientras se está desarrollando la aplicación *Android*, para ello se debe configurar de la siguiente forma.

Primero, dentro de *Settings -> Security* en el terminal se debe activar la opción *Unknown sources* que permite la instalación de aplicaciones de otras fuentes distintas al *Play Store de Google,* en este caso provendrán del entorno de trabajo sobre el que desarrollamos.

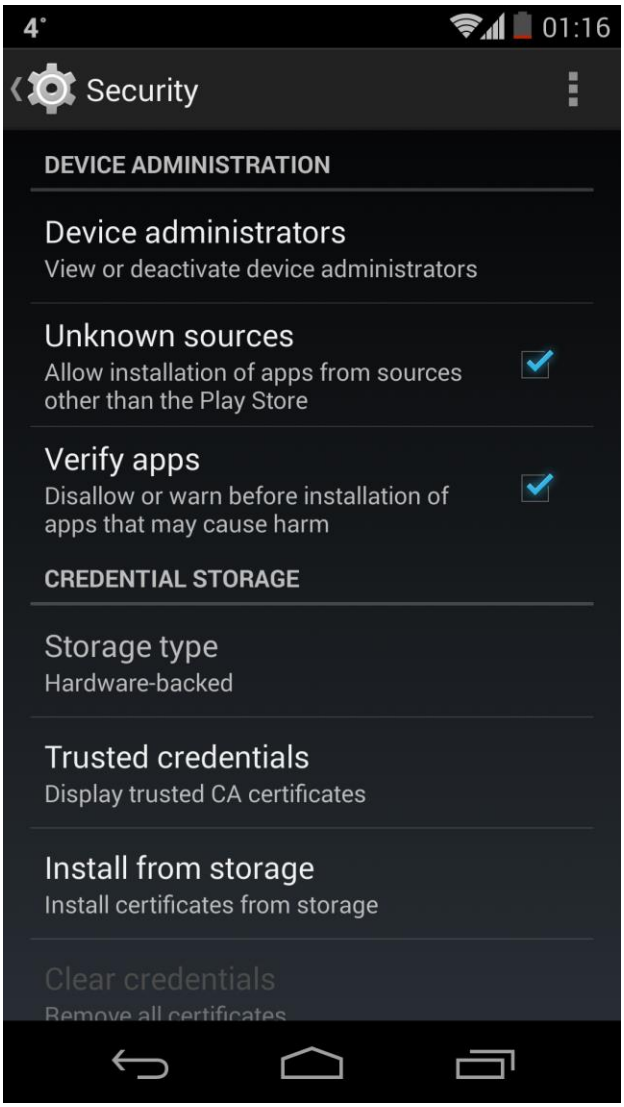


Figura 60. Sección de Seguridad dentro de la configuración del terminal Android

Segundo, se deben habilitar las opciones de desarrollador del terminal. Esto se consigue seleccionando repetidamente, sobre 10 veces, el texto *Build Number* dentro de *Settings -> About Phone*. Cuando se haya habilitado esta opción se mostrará un mensaje en la pantalla confirmando este cambio.

Una vez hecho esto se puede acceder a un nuevo conjunto de configuraciones dentro de los *Settings,* los *Developer Options*. Hay que prestar especial atención a que se tenga seleccionada la opción de *USB Debugging*, con la cual podremos ejecutar y depurar la aplicación Android lanzándola directamente desde nuestro entorno de *Eclipse* conectando el terminal con nuestro entorno mediante un cable *USB*.

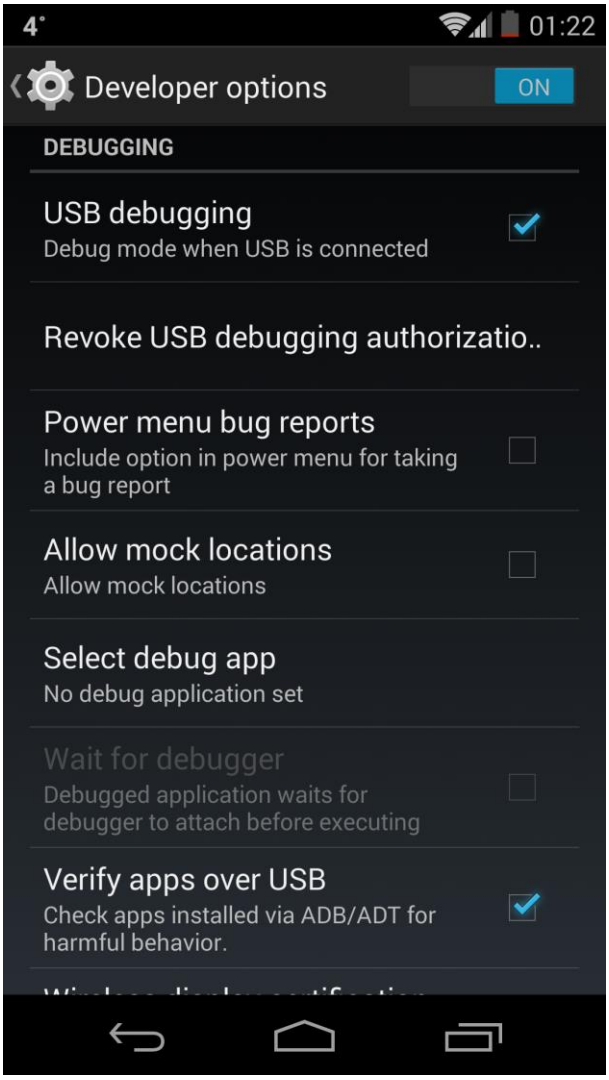


Figura 61. Sección para desarrolladores en un terminal Android

VI. 7. Plugin de *GITHUB* para *MAC*

El sistema de control de versiones elegido es GIT debido a su versatilidad y funcionalidad frente a otros sistemas. El repositorio a utilizar será *Github* y para su uso desde nuestro entorno de trabajo se pueden seguir diferentes caminos:

- Instalación del software que se proporciona en su *site* principal con el cual, y mediante línea de comandos, se puede gestionar el código alojado tanto en nuestro entorno de trabajo como en el servidor Git.

- Instalación del plugin para *Eclipse* con el que integraremos las funcionalidades de repositorio *Github* directamente en nuestro entorno de *Eclipse*. Con ello tendremos acceso directo a las funciones necesarias para la sincronización de nuestro repositorio local con el compartido.

- Instalación de la aplicación nativa para nuestro sistema que proporciona un entorno visual más amigable e intuitivo que la ejecución de comandos mediante un terminal y mayor control y visión que el plugin para *Eclipse*.

Para este proyecto de fin de carrera se ha elegido esta última opción por el control y manejabilidad que la aplicación llega a ofrecer.

Lo primero que es necesario configurar para poder empezar a usar *Github* es el nuevo repositorio donde alojar el código. Desde el perfil del usuario y dentro de la sección *Repositories* se tiene acceso a la funcionalidad para crear uno nuevo.

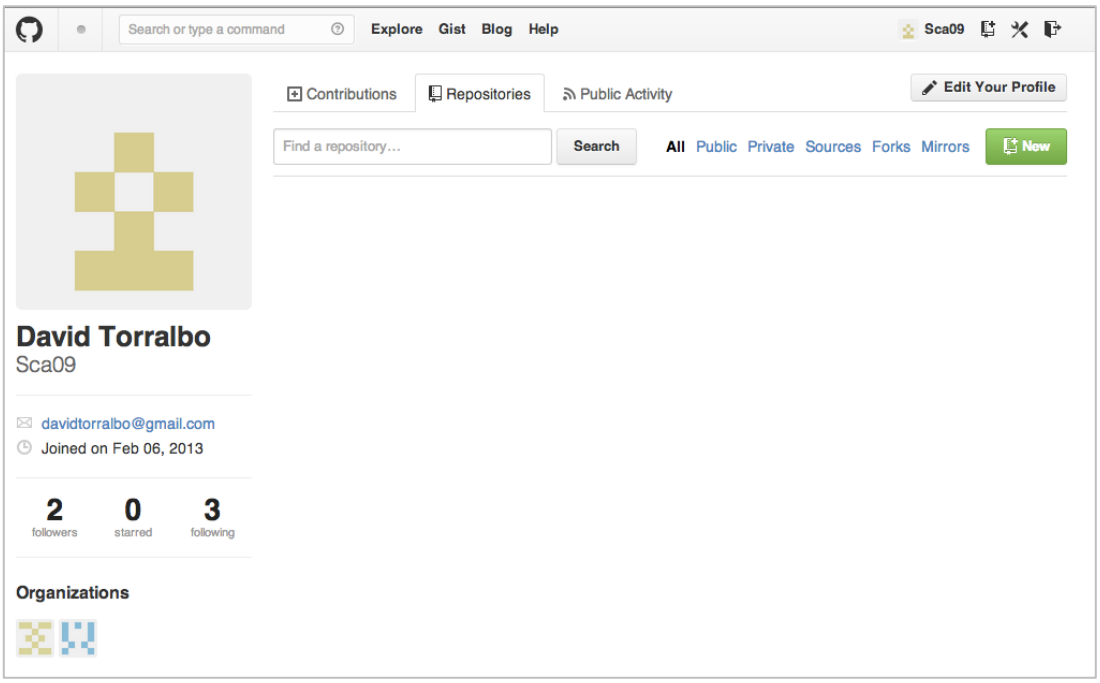


Figura 62. Pantalla principal en GITHUB

Se puede entonces configurar distintas variables del nuevo repositorio.

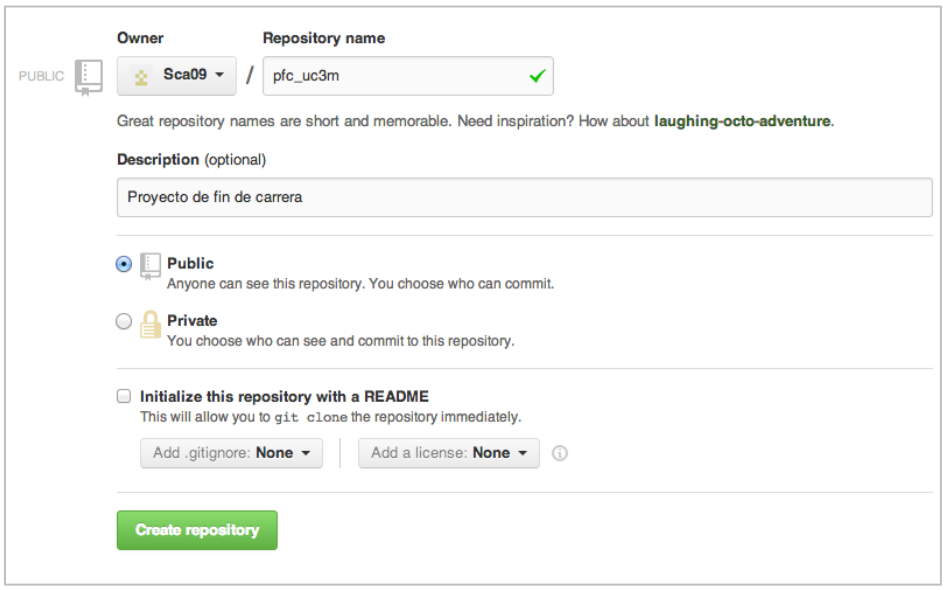


Figura 63. Creación de un nuevo proyecto en GITHUB

En esta pantalla se puede especificar el nombre del repositorio así como la entidad administradora del mismo. Si el repositorio será de acceso público o se necesitará garantizar el acceso a los usuarios para que puedan acceder al mismo, etc.

Una vez creado tendremos los datos necesarios para poder acceder desde nuestro entorno de trabajo al repositorio y así hacer una copia sobre la que trabajar y sincronizar el desarrollo que se vaya haciendo.

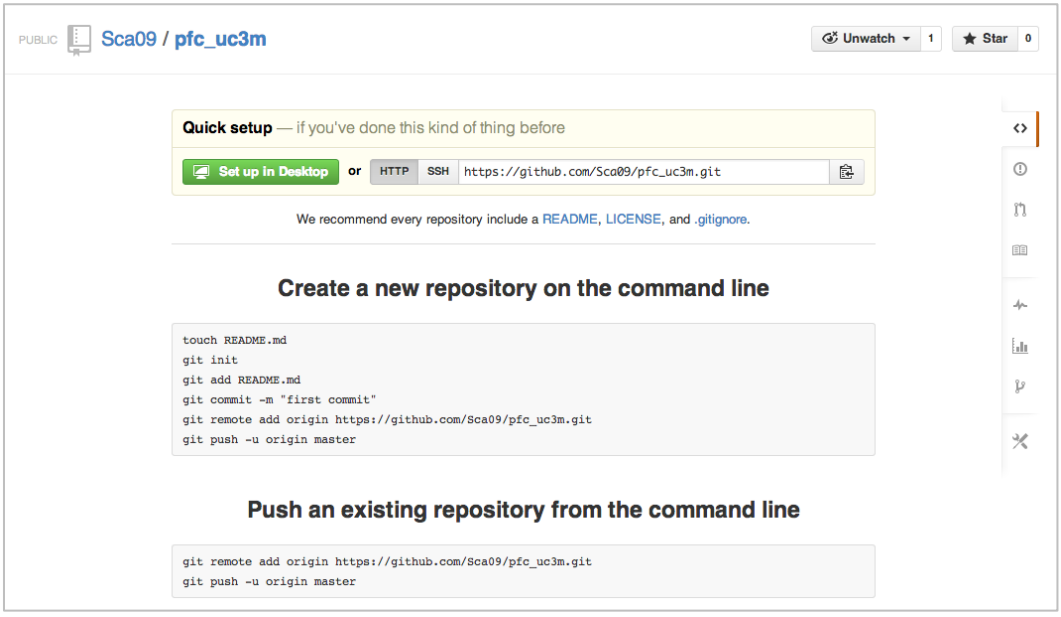


Figura 64. Página inicial de un nuevo proyecto

En el momento de la creación del nuevo repositorio, se tendrán disponibles dos direcciones con las que es posible clonar dicho repositorio en nuestro entorno de trabajo:

- *HTTP: https://github.com/Sca09/pfc_uc3m.git*
- *SSH: git@github.com:Sca09/pfc_uc3m.git*

Igualmente si se tiene la aplicación nativa instalada, se puede hacer una instalación más sencilla seleccionando el botón "*Set up in Desktop*" cuya acción lanzará automáticamente dicha aplicación. La instalación de la aplicación nativa se basa en la descompresión del paquete descargado. La configuración del usuario a utilizar se encontrará vacía, por lo que es necesario acceder con el nombre de usuario y contraseña del usuario deseado. Para ello hay que abrir las opciones en *GitHub -> Preferences*.

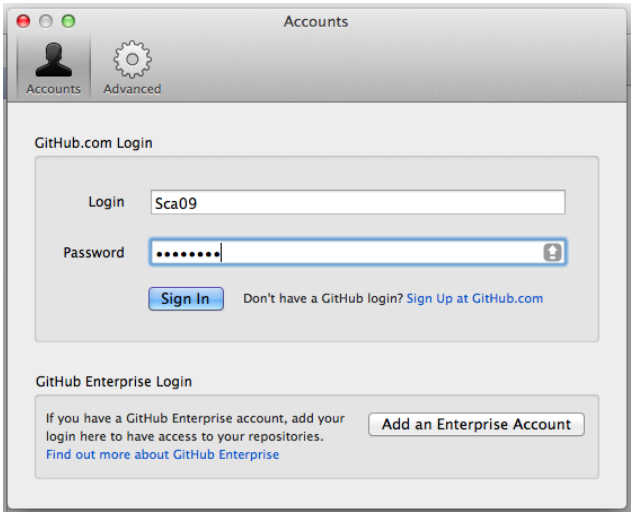


Figura 65. Pantalla de identificación de la aplicación GITHUB

En cuanto el usuario se ha iniciado sesión con sus credenciales sus repositorios aparecen como accesibles desde la herramienta. Desde los cuales podremos gestionar nuestro código.

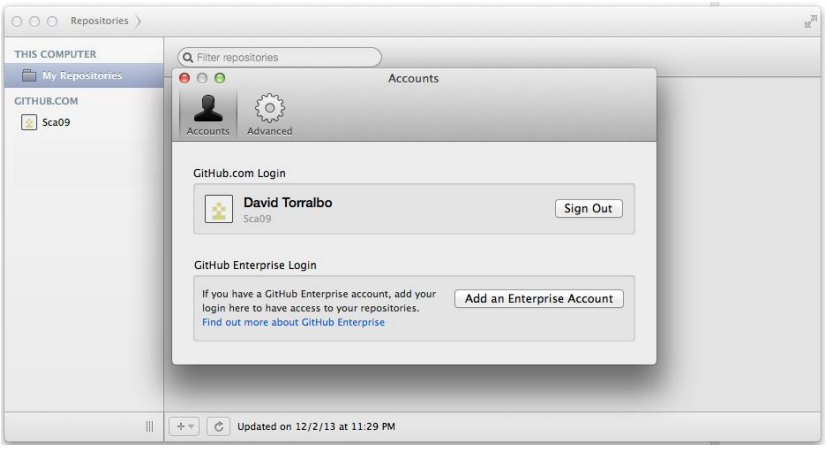


Figura 66. Identificación de usuario en GITHUB

Si seleccionamos desde la página de *Github* en la acción de "*Set up in Desktop*" se iniciará el proceso de sincronización del repositorio seleccionado. Para terminar la clonación del repositorio solo es necesario especificar la ruta deseada y directamente se creará el vínculo entre ambos entornos.

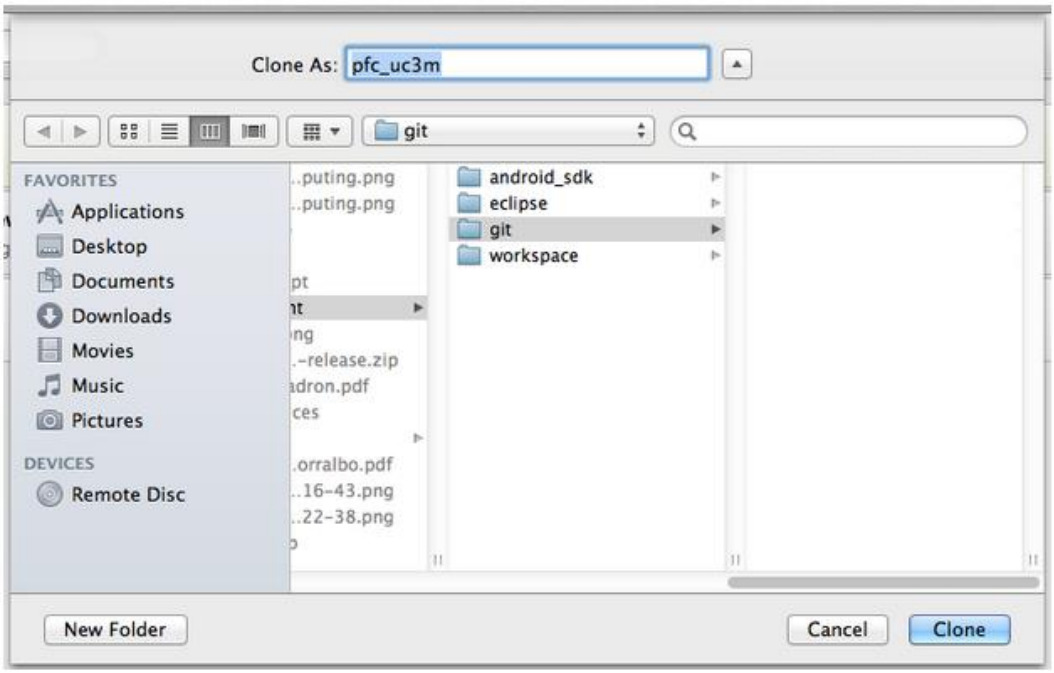


Figura 67. Selección de la ruta local donde clonar el repositorio remoto

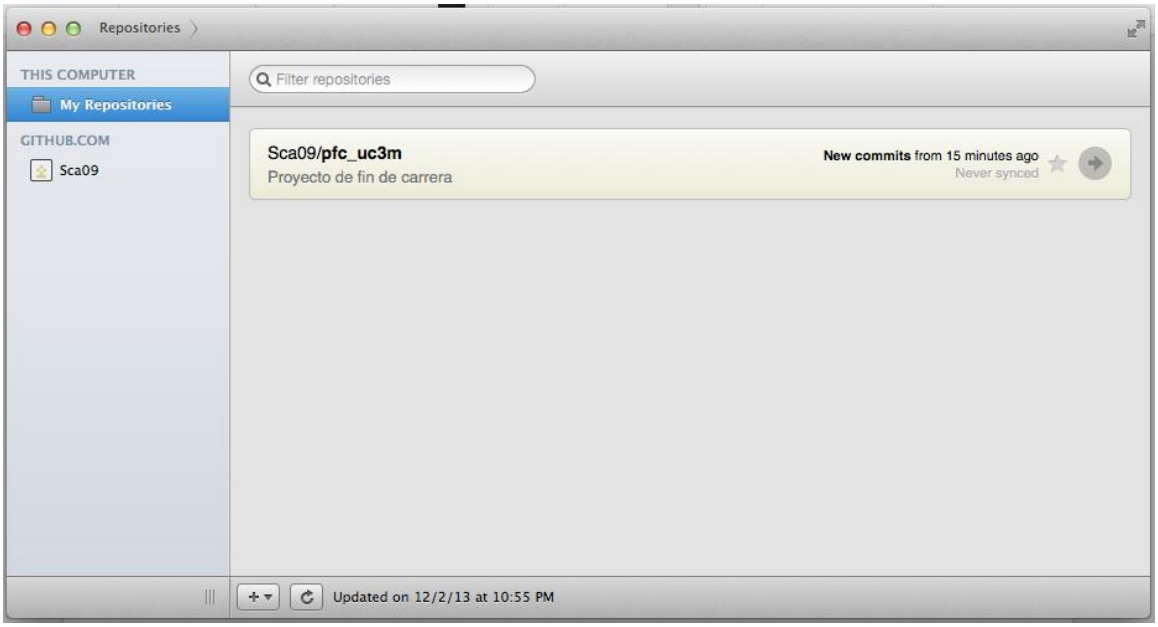


Figura 68. Repositorio remoto clonado al entorno de trabajo local

Este proceso de clonado de un repositorio puede hacerse también directamente desde esta aplicación nativa una vez el usuario está correctamente identificado. Para ello hay que seleccionar la entidad en la que se encuentra el repositorio y seleccionar la opción de clonar a nuestro entorno.

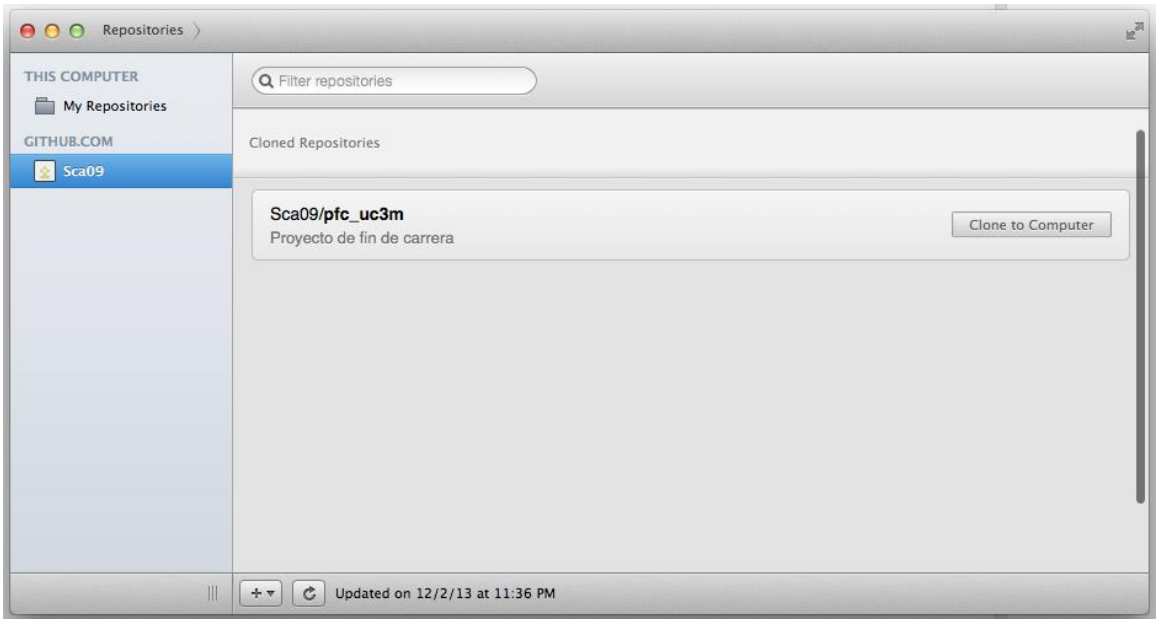


Figura 69. Clonado de un repositorio remoto asociado a un usuario

Una vez que el repositorio ha sido clonado, ya estará listo para poder ser utilizado desde nuestro entorno local de trabajo.

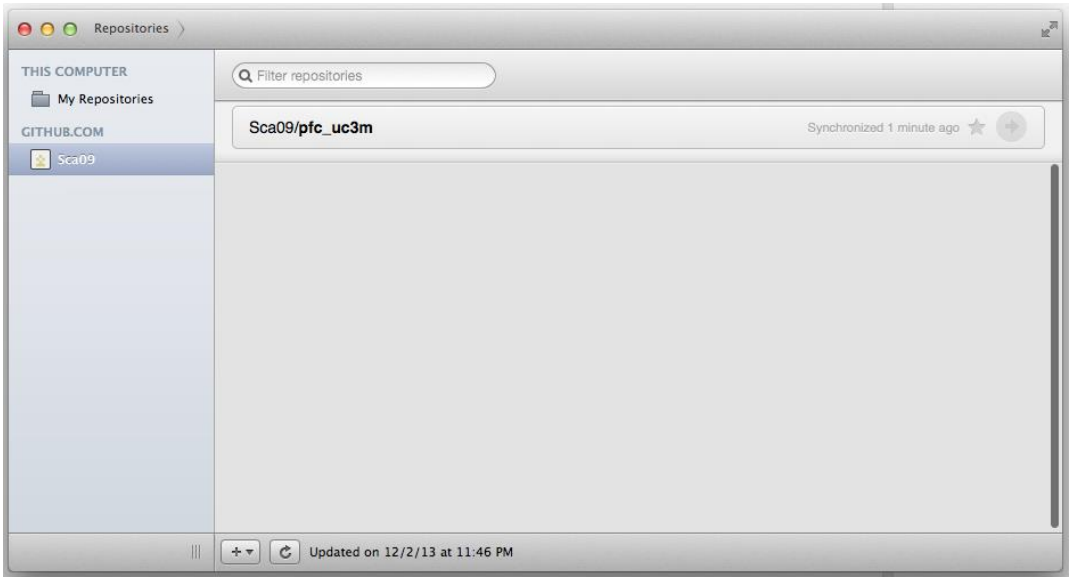


Figura 70. Repositorio remoto clonado

Gracias a la aplicación nativa se pueden comparar y sincronizar los cambios realizados, pudiéndose publicar al repositorio remoto. La aplicación muestra los cambios que se van produciendo desde la última sincronización.

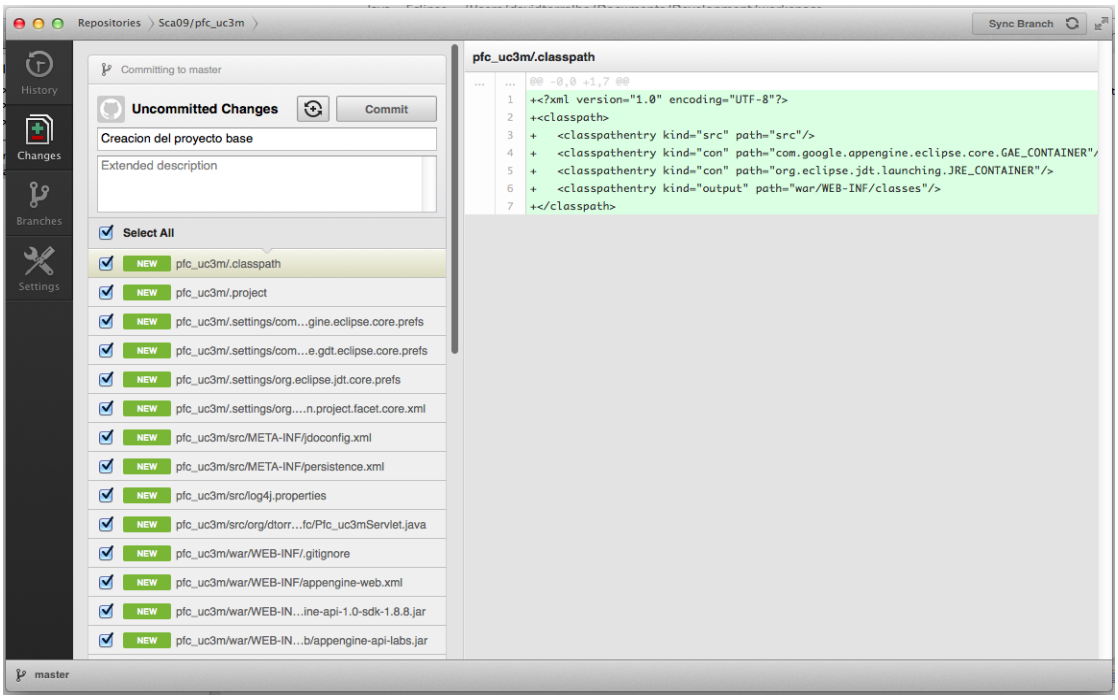


Figura 71. Vista de los cambios pendientes de ser subidos al repositorio remoto

En esta misma aplicación se puede realizar funciones como *commit* que encapsula los cambios que se seleccionen para ser publicados.

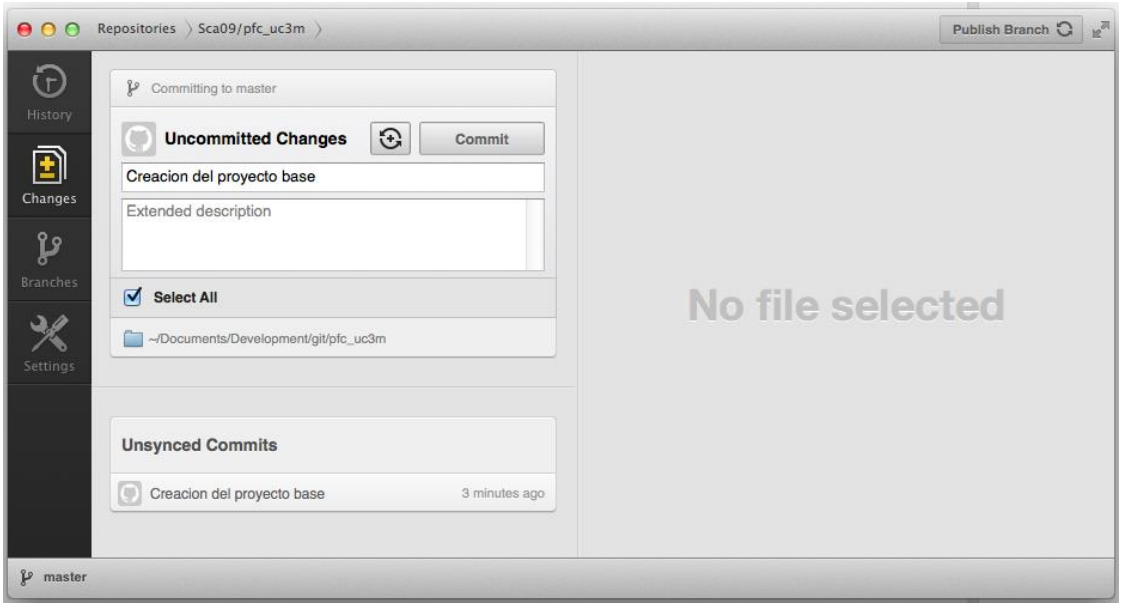


Figura 72. Proceso de commit mediante la aplicación GITHUB

Una vez hecha esta publicación los repositorios locales y distribuidos se encuentran sincronizados. A partir de ahora los cambios que queramos publicar desde el entorno local, después de haber ejecutado el *commit*, serán sincronizados simplemente seleccionando el botón *Sync* junto al paquete de cambios deseado.

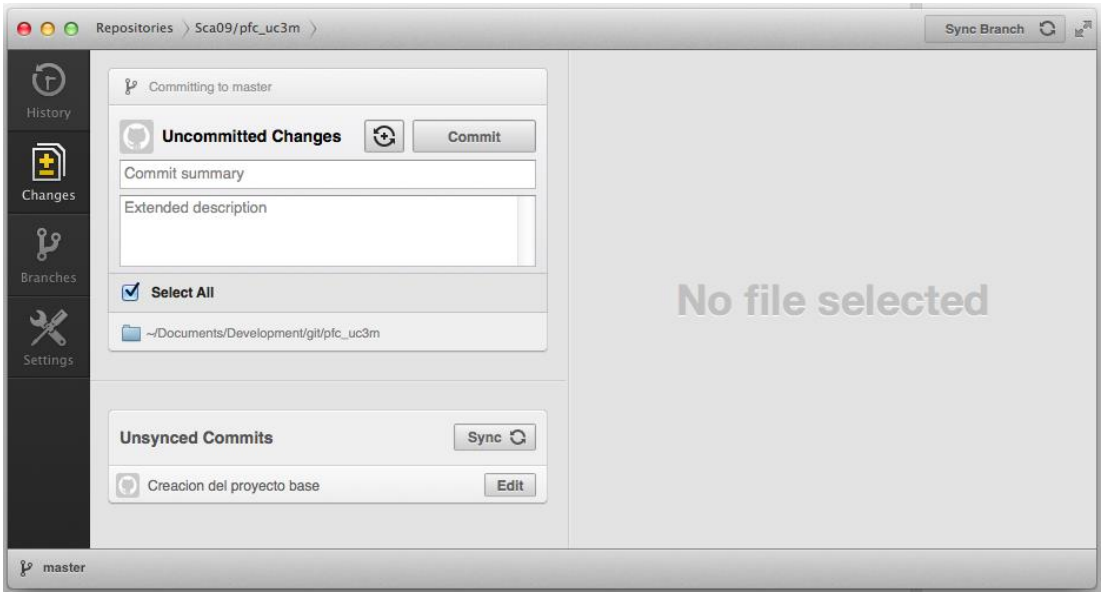


Figura 73. Proceso de push mediante la aplicación GITHUB

VI. 8. *Firefox* y *Firebug Tool* para *Firefox*

Para este proyecto es posible utilizar cualquier navegador que el mercado ofrece, pero se ha utilizado *Firefox* con el fin de aprovechar las funcionalidades que su *plugin Firebug* ofrece.

Este *plugin* es de gran utilidad a la hora de desarrollar sobre lenguaje *HTML*. Para este proyecto es utilizado con el fin de descubrir los servicios o acciones que nuestra API creada puede ofrecer. Su consola de comandos aporta la funcionalidad de autocompletado que irá mostrando dichos servicios previamente cargados.

El navegador *Firefox* puede ser descargado desde la página que *Mozilla* tiene destinada para este servicio. Se ha utilizado su última versión disponible a la hora de realizar el estudio, esta es 25.0.1.

Una vez el navegador se encuentra instalado, es posible encontrar *Firebug* dentro de su herramienta para desarrolladores, en *Tools -> Web Developer -> Get More Tools* o accediendo directamente a su página principal.

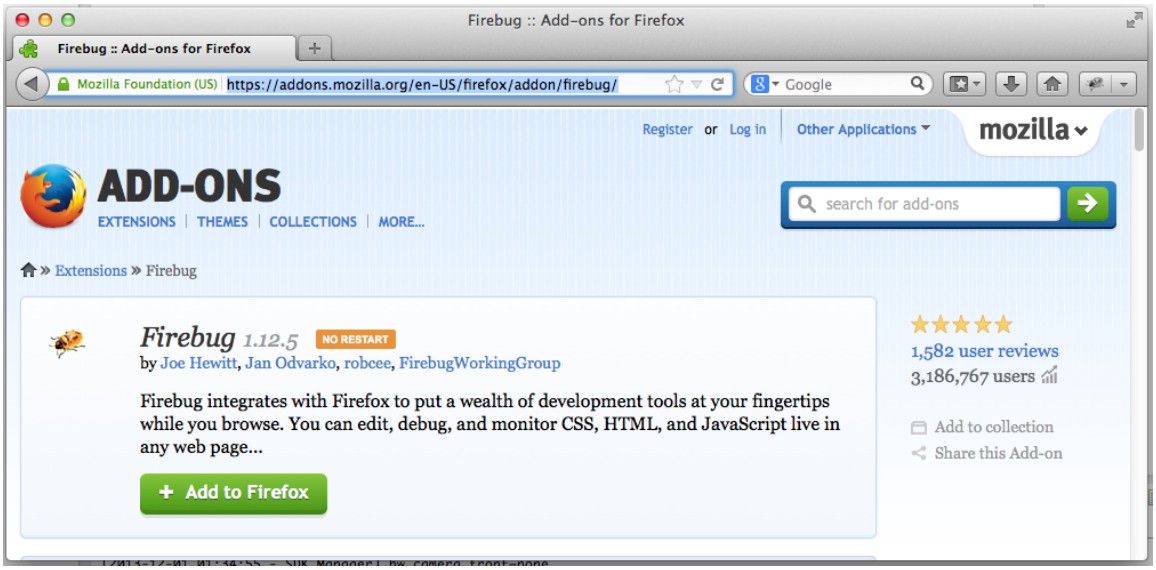


Figura 74. Sección de add-ons de Mozilla Firefox

Una vez seleccionado e instalado, las nuevas funcionalidades están disponibles bajo la consola que el *plugin* ofrece.

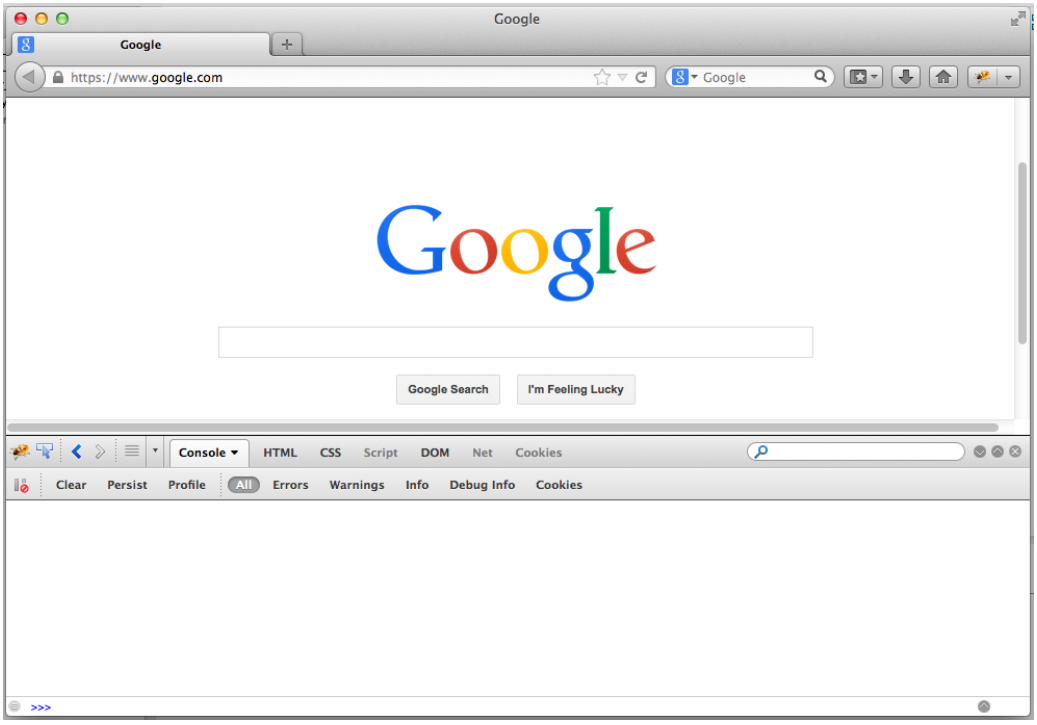


Figura 75. Consola de Firebug

Esta consola es de ayuda a la hora de desarrollar nuestra aplicación en formato HTML. Su función de autocompletado puede orientar al desarrollador entre los servicios que pueden ser usados.

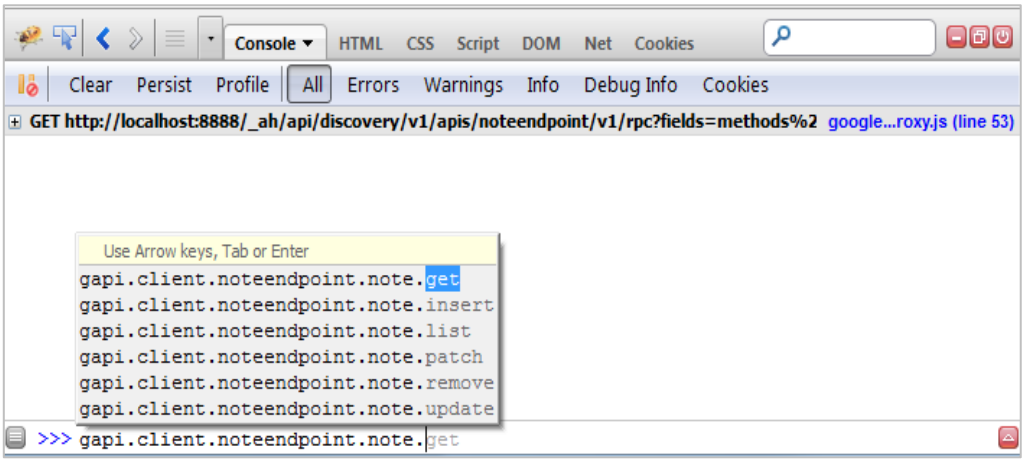


Figura 76. Función de autocompletado del a consola de Firebug

VI. 9. *RESTClient*

Esta herramienta es de gran utilidad y ayuda para conectarse a servicios de tipo *RESTFul*. De una forma gráfica muy amigable encapsula la llamada a realizar facilitando la maquetación de los parámetros a enviar en la llamada, así como otros parámetros como cabeceras o incluso los modos de llamad, así como la visualización de la respuesta obtenida del servidor *REST*.

Herramienta *open source* que puede ser descargada desde su página principal en la que se puede encontrar información adicional de utilidad así como tutoriales de uso. Para este proyecto se ha usado su versión 3.2.1. para *MAC*.

La simple descompresión del paquete descargado es suficiente para poder hacer uso de esta herramienta. No es necesaria ninguna especial configuración para que la herramienta esté preparada para su uso.

En el desarrollo de nuestra lógica de negocio se hará uso de sus funcionalidades para comprobar de una forma alternativa que nuestros servicios funcionan correctamente y la respuesta obtenida es la deseada. Además se demostrará como los servicios creados gracias a la funcionalidad de los *Google Cloud Endpoints* pueden ser igualmente accedidos como si de una comunicación *RESTFul* se tratase, abriendo así a más posibilidades de uso de nuestro servicio.

VI. 10. *Google App Engine Tool*

Con este servicio que *Google* ofrece se pueden crear servidores de aplicaciones sobre los que correrán la lógica de negocio que la aplicación necesita. Con unos simples pasos se puede disponer de un servicio completamente configurado y accesible públicamente.

La configuración de estos servidores se hace desde la página principal de *App Engine* en la que se muestran las diferentes aplicaciones existentes creadas o asociadas a nuestro usuario de *Google*.

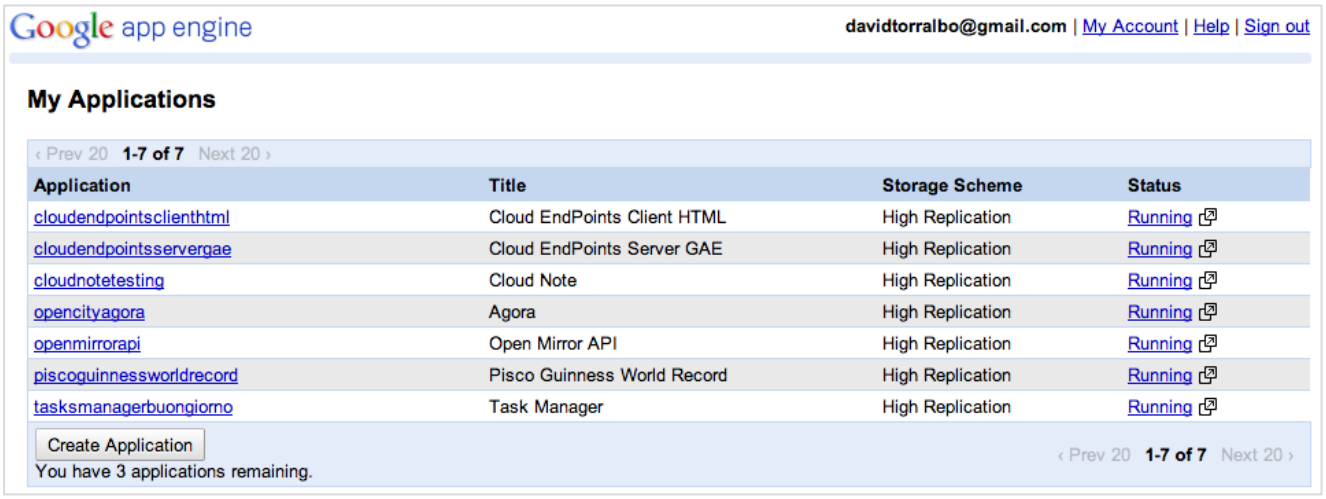


Figura 77. Página de configuración de aplicaciones de App Engine

Para la creación de una nueva aplicación se deben rellenar los siguientes datos que servirán de configuración de los entornos de trabajo:

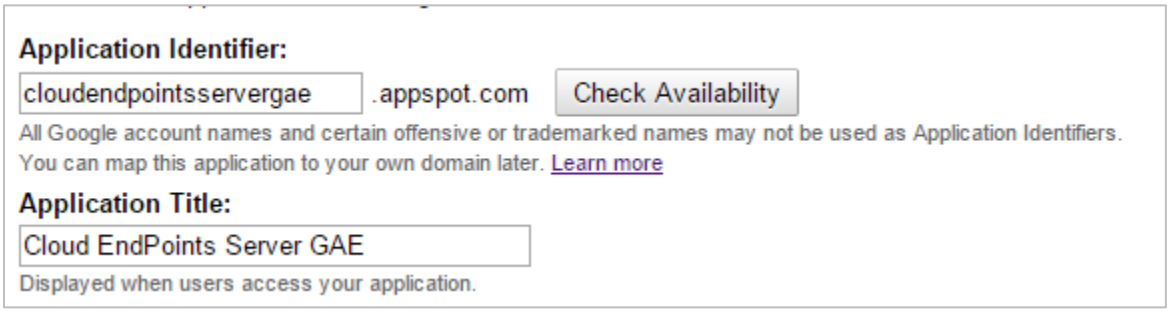


Figura 78. Creación de una nueva aplicación en *App Engine*

Se debe elegir un identificador de la aplicación que compondrá la dirección estándar para acceder al servicio concatenando este identificador al dominio "*appspot.com*". Se define igualmente el título de la aplicación y por último el posible mecanismo de autenticación de los usuarios que accedan a la aplicación.

Para este proyecto de fin de carrera se ha utilizado la opción de autenticación mediante una cuenta Google. También es posible configurar la autenticación de la aplicación mediantes cuentas de aplicaciones Google como por ejemplo "*alumnos.uc3m.es*". Y finalmente es posible abrir la autenticación a cualquier cuenta que sea controlada por un servidor OpenID tales como *Yahoo, PayPal o MySpace*.

Una vez creada la aplicación, se tiene acceso al panel principal de configuración en el que podremos modificar los parámetros de nuestra instancia y comprobar el estado de nuestros servidores en uso. Así por ejemplo se puede ver cuántos servidores activos se encuentran en cada momento y sus detalles.

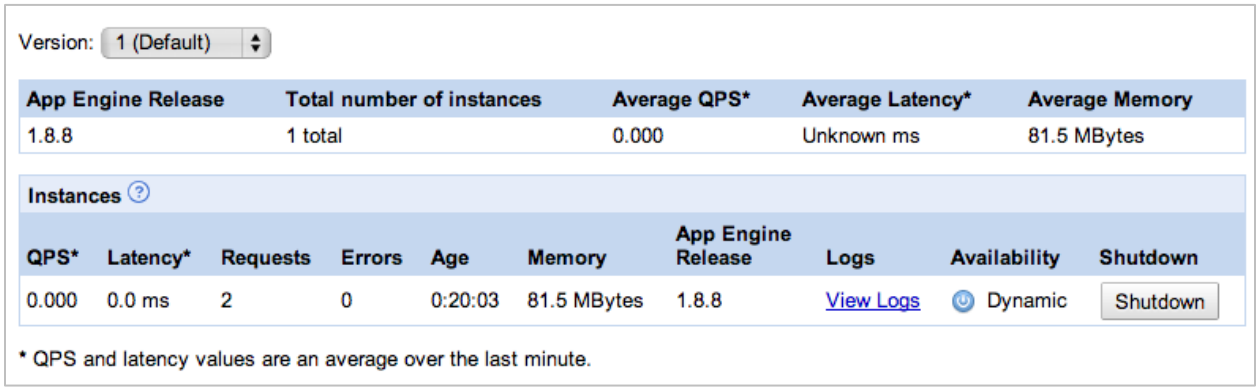


Figura 79. Panel de información de la aplicación

Se pueden acceder a su apartado de *logs* para comprobar las llamadas que se han recibido y ver el comportamiento de nuestra aplicación antes las peticiones realizadas.

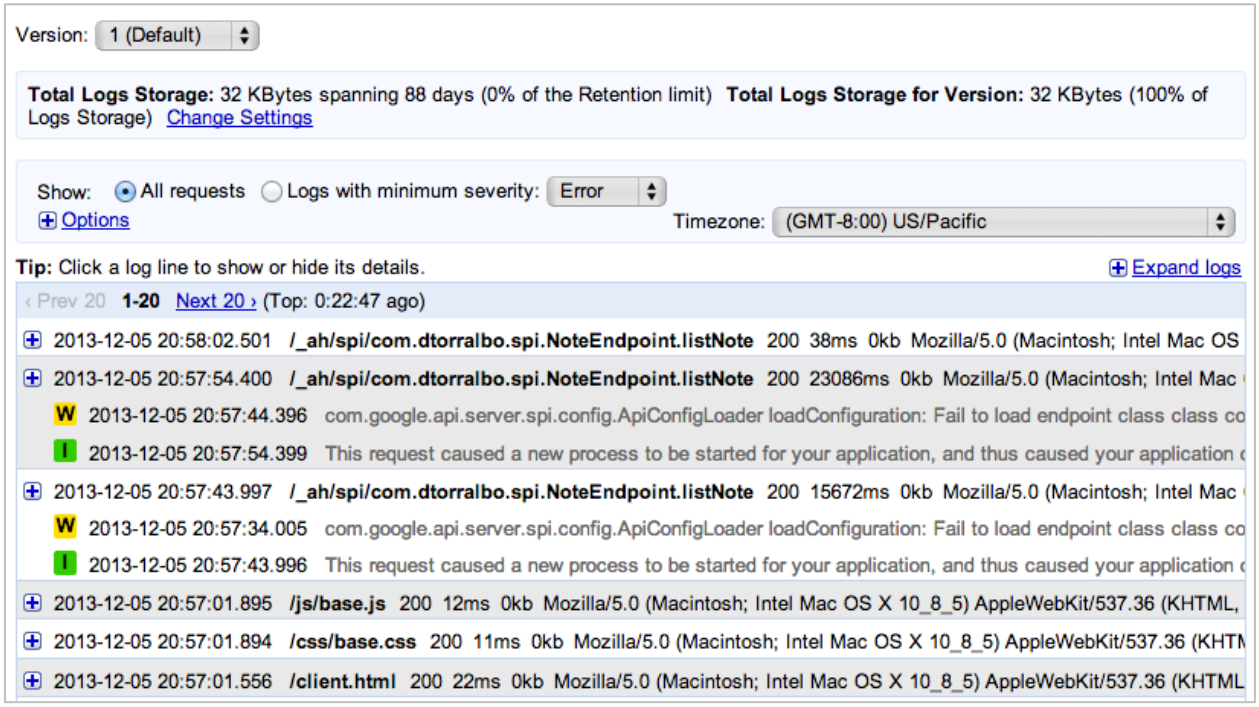


Figura 80. Panel de logs en App Engine

Otra sección importante es la de la gestión de versiones en la que se puede controlar las diferentes versiones desplegadas en el servidor. Así se podría configurar como versión por defecto cualquiera de las desplegadas, no teniendo que ser necesariamente la última. Cada versión provee de la dirección necesaria para ser lanzada igualmente.

	Version	Default	Deployed	Delete
○	10 instances | 44.87 MBytes | java	No	315 days, 7:41:09 ago by davidtorralbo@gmail.com	Delete
○	11 instances | 44.94 MBytes | java	No	314 days, 6:10:13 ago by davidtorralbo@gmail.com	Delete
○	12 instances | 45.94 MBytes | java	No	308 days, 7:35:36 ago by davidtorralbo@gmail.com	Delete
○	13 instances | 48.18 MBytes | java	No	300 days, 10:22:57 ago by davidtorralbo@gmail.com	Delete
○	14 instances | 48.21 MBytes | java	No	297 days, 7:10:03 ago by davidtorralbo@gmail.com	Delete
◉	15 instances | 46.24 MBytes | java	**Yes**	294 days, 12:12:04 ago by davidtorralbo@gmail.com	*Cannot delete default version.*

Make Default

Figura 81. Listado de versiones de la aplicación desplegadas en App Engine

A través de esta herramienta se puede acceder a los datos almacenados en la base de datos para su gestión manual.

Query | Create

By kind: Note kinds as of 0:00:44 ago — Number of Columns to Display: 100

Options

Note Entities

‹ Prev 20 **1-3** Next 20 ›

ID/Name	description	emailAddress
name=id1	note #1.1	davidtorralbo@gmail.com
name=id2	note #2	100030474@alumnos.uc3m.es
name=id3	note #3	davidtorralbo@gmail.com

Delete | Flush Memcache ‹ Prev 20 **1-3** Next 20 ›

Figura 82. Vista de la base de datos asociada a la aplicación en App Engine

Esta herramienta es por lo tanto el panel de configuración del servidor central que proporciona la lógica de negocio de la aplicación.

VI. 11. *Google API Explorer*

El servicio *Google Cloud EndPoints* no solo tiene la función de generar los clientes con los que se puede acceder a la lógica de negocio desde las diferentes aplicaciones a crear, sino que además genera los puntos de acceso para estos clientes. Esto se trata por parte de la plataforma de *Google* como si de una *API* se tratase, por lo que es posible ayudarse del *Google API Explorer* para acceder a este *API* y probar su correcto funcionamiento.

El acceso al mismo debe hacerse mediante la *URI /_ah/api/explorer* de nuestro dominio. Esta *URL* completa nos da acceso a las API declaradas bajo el dominio especificado.

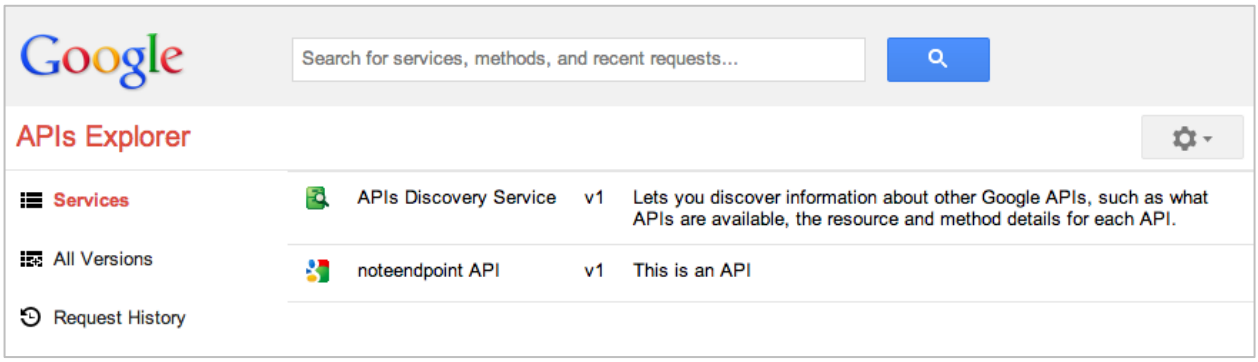


Figura 83. Vista principal de la herramienta API Explorer de Google

Se puede ir navegando entre las APIs y los servicios ofrecidos en cada caso para explorar todas las opciones disponibles.

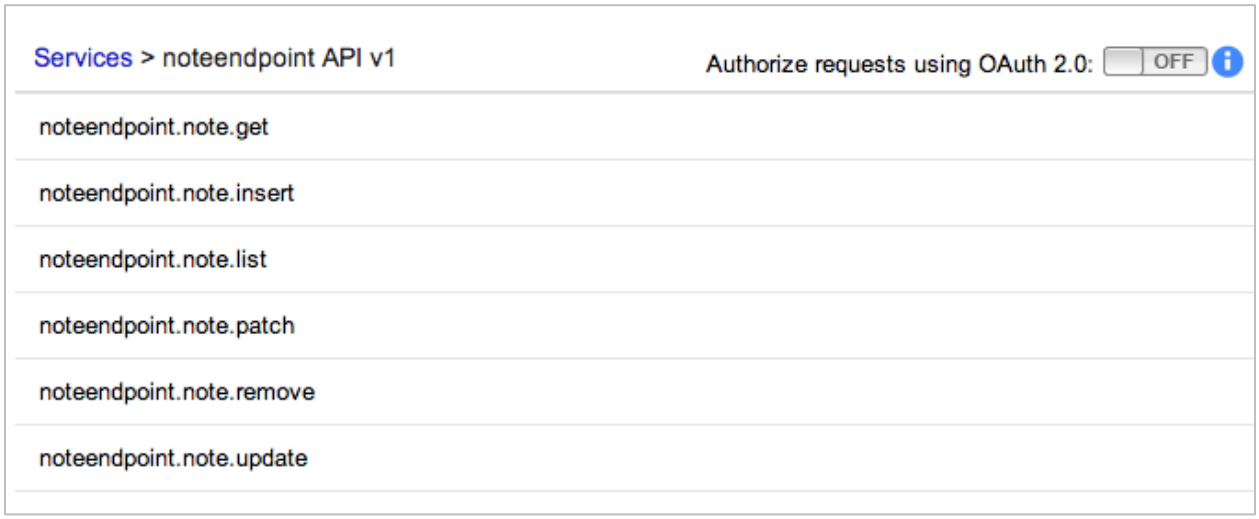


Figura 84. Listado de servicios disponibles por la aplicación

Y para cualquiera de los servicios presentados se puede ejecutar la correspondiente petición a nuestra API final con la asistencia de esta herramienta. Es de gran ayuda ya que presenta un formulario con los parámetros que la llamada requiere en cada caso y muestra la ejecución de la petición así como el resultado devuelto por la *API*.

Es posible por lo tanto realizar una petición desde esta herramienta, y así comprobar los detalles de la petición realizada.

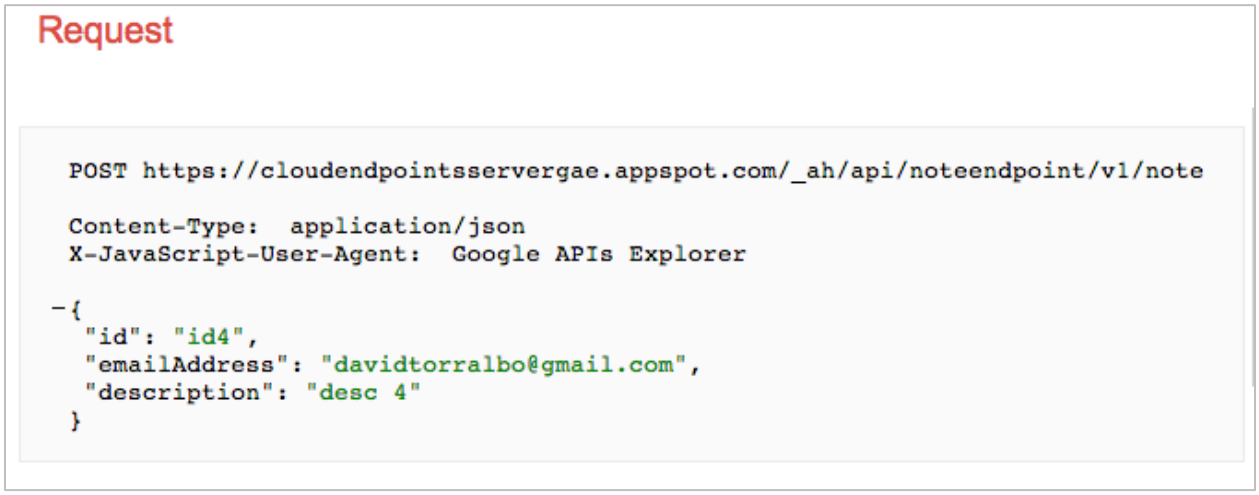


Figura 85. Ejemplo de una petición realizada a través de la herramienta API Explorer

También es posible ver la respuesta completa a la petición realizada.

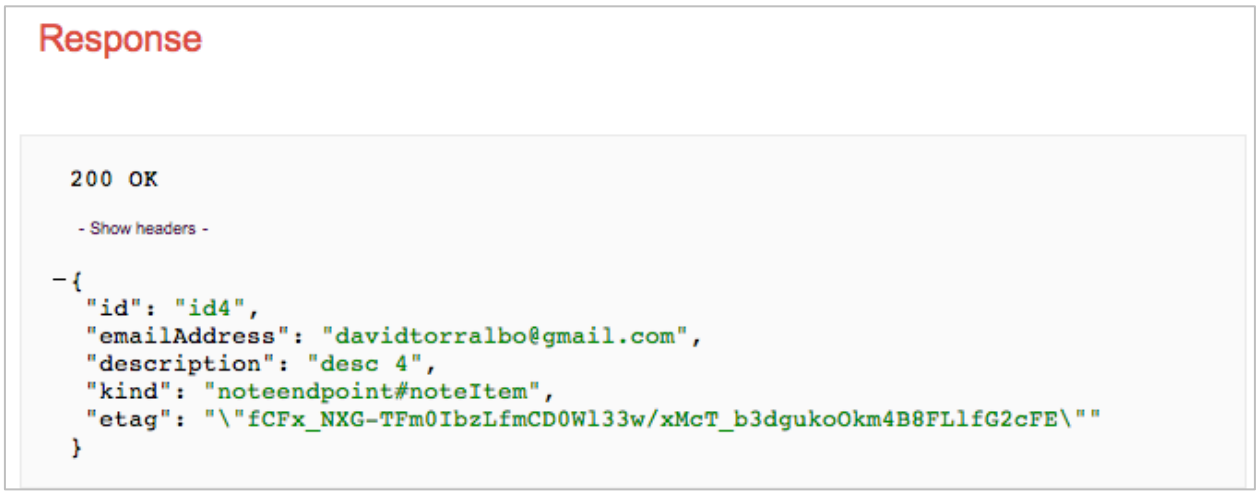


Figura 86. Respuesta a la petición realizada

VI. 12. *Google Drive*

Este servicio proporcionado por *Google* ha sido de gran importancia en la elaboración de este proyecto de fin de carrera. El almacenamiento de documentos en la red así como la posibilidad de compartir ficheros entre distintos usuarios y la posibilidad de editarlos y comentarlos por cualquiera de los implicados ha simplificado en gran medida la posibilidad de la realización del proyecto en remoto.

En todo momento, tanto el tutor como el alumno han mantenido toda la información y los documentos actualizados encontrando un único punto común accesible desde cualquier dispositivo. Con esto se ha evitado los problemas de desincronización de documentos entre alumno y tutor así como una rápida comunicación entre ambas partes. Cualquier documento creado y almacenado por alguno de los usuarios puede ser compartido con otros usuarios.

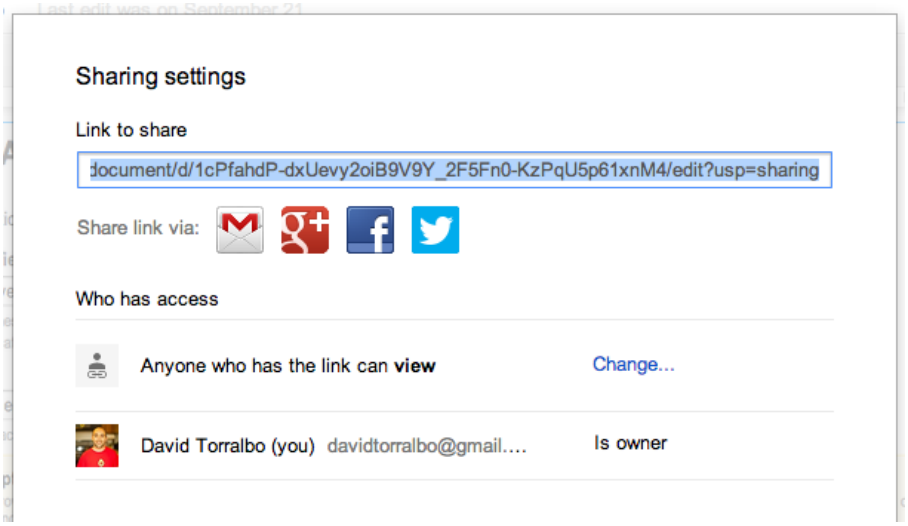


Figura 87. Panel de configuración de los permisos al compartir un documento en Google Drive

En este momento quienes cumplan los requisitos especificados en la configuración de compartir el documento tendrá acceso al mismo. Este acceso puede ser también restringido a las acciones permitidas, tales como solo comentar, solo visualizar o con privilegios de edición del documento.

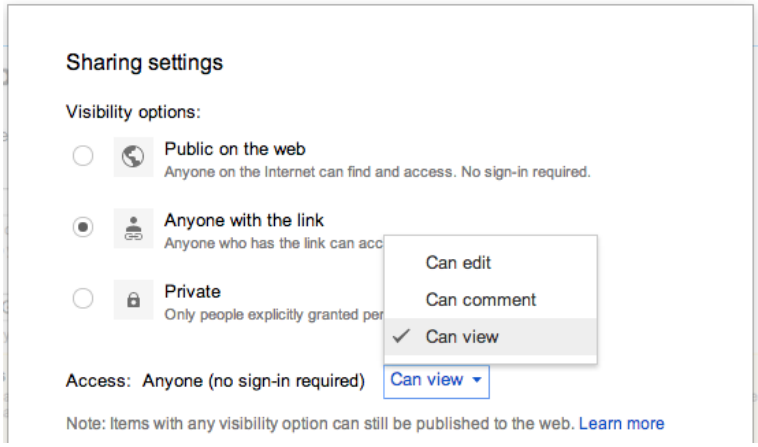


Figura 88. Configuración de las acciones y de los roles que pueden acceder a cada documento

Así los documentos pueden ser configurados como totalmente públicos, o solo accesibles por los usuarios que posean la url del documento o incluso como un documento privado cerrado únicamente a los usuarios añadidos a la lista de usuarios permitidos. En este último caso, cada usuario puede configurarse con diferentes privilegios sobre el mismo documento.

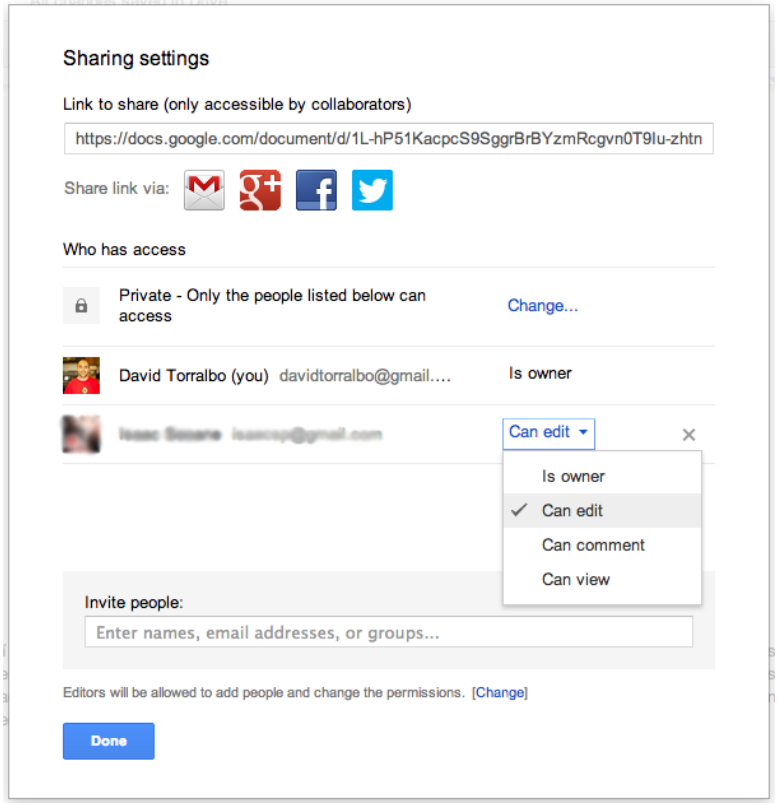


Figura 89. Configuración de los permisos por cada usuario invitado

Una de las herramientas que *Google Drive* proporciona y que de mayor utiliza ha resultado son los comentarios, con los que apuntar y remarcar posibles puntos de conflicto y revisión.

Figura 90. Comentarios en documentos en Google Drive